Czech Technical University in Prague

Faculty of Nuclear Sciences and Physical Engineering

DISSERTATION THESIS

Distribuovaná správa dat v experimentech na RHIC a LHC

Michal Zerola

Distributed Data Management in Experiments at RHIC and LHC

DEPARTMENT OF MATHEMATICS

2012

Název:	**Distribuovaná správa dat v experimentech na RHIC a LHC**
Autor:	Mgr. Michal Zerola
Obor:	matematické inženýrství
Druh práce:	Disertační práce
Vedoucí práce:	doc. Michal Šumbera, CSc., DSc., Ústav jaderné fyziky, Akademie věd ČR Dr. Jérôme Lauret, STAR experiment, Brookhaven National Laboratory, USA
Konzultant:	Doc. RNDr. Roman Barták, Ph.D., Katedra teoretické informatiky a matematické logiky, Matematicko-fyzikální fakulta, Univerzita Karlova v Praze
Abstrakt:	Táto dizertačná práca pojednáva o skutočných potrebách prenosu dát jedného z najväčších bežiacich fyzikálnych experimentov na svete. Obsahuje teoretické štúdie a prezentuje vývoj riešiaceho modelu založeného na podmienkach. Praktická časť sa skladá z návrhu architektúry, meraní a vyhodnotenia výkonnosti automatického plánovacieho systému. Nad riešiacimi technikami dátových prenosov boli tiež vytvorené ich deriváty, ktoré boli aplikované i v oblasti robotiky a sú prezentované v záverečnej prílohe.
Klíčová slova:	plánování, grid, přenos dat, programování s omezujícími podmínkami, celočíselné programování

Title:	**Distributed Data Management in Experiments at RHIC and LHC**
Author:	Mgr. Michal Zerola
Advisor:	doc. Michal Šumbera, CSc., DSc., Nuclear Physics Institute, ASCR Dr. Jérôme Lauret, STAR experiment, Brookhaven National Laboratory, USA
Consultant:	Doc. RNDr. Roman Barták, Ph.D., Dept. of Theoretical Computer Science and Math. Logic, Faculty of Mathematics and Physics, Charles University in Prague
Abstract:	This thesis discusses the real life data transfer and placement needs of one of the largest physics experiments in the world. It inheres the theoretical studies of the underlying problem and presents the evolution of the constraint-based solving model. Practical part consists of the architecture design, measurements and performance evaluation of the automated planning system. Derived techniques from data transfers were applied also in the field of robotics and are discussed in the appendix.
Keywords:	planning, Grid, data transfer, constraint programming, integer programming

Contents

Acknowledgements

Because work included in this dissertation thesis is the final outcome of a synergy with many STAR colleagues I would like to initially thank to people involved in this work in a way or another. At the same time I would like to apologize to those who were important in completing this thesis and I could not mention them personally line by line.

First and foremost I offer my sincerest gratitude to each of my supervisors, Dr. Michal Šumbera, Dr. Jérôme Lauret, and Dr. Roman Barták. They all showed me their shared enthusiasm and passion on the research, while each of them was providing me with a specific guidance, expertise and inspiration. As a result, research life became smooth and rewarding for me. Having such professionals covering diverse, but still overlapping fields of science, is often a dream for a student, moreover if their balanced attitude passes beyond the engagements and turns into care. One simply could not wish for better supervisors; and it has been a honor and pleasure for me to work with them.

I was lucky to meet several great colleagues from STAR Collaboration, discuss with them not only work-related topics and spend a lot of free time during my stays at Brookhaven National Laboratory. My co-workers and great friends from the Prague's heavy-ions group have been definitely a fundamental element and support in physics and computing related discussions along a lot of fun we have gone through together. My great thank goes especially to Jana & Jaroslav Bielčík, Peter Chaloupka, Jan Kapitán, Michal Bysterský, and Pavel Jakl. My friend and colleague from Institute of Computer Science Stanislav Slušný deserves another big thank for the help and perfect time we had during the work on our common papers in robotics.

Last but definitely not least, there is my family I want to thank to, without which none of this work would be possible. My parents, grand parents, and sister provided me constant support, encouragement and help all the time and stayed by me especially when it was most needed.

Declaration of Originality

This doctoral thesis contains results of my research carried out at the Nuclear Physics Institute between years 2007 and 2011. Most of this work was carried out within STAR Collaboration. Excluding introductory parts, the research described in this thesis is original unless where an explicit reference is made to work of others. I further state that no part of this thesis or any substantially the same has been submitted for any qualification other than the degree of Doctor of Philosophy at the Czech Technical University in Prague.

Glossary

cache policy heuristic used to select the entry to eject with the regard to cache space. 36

cluster group of closely linked computers, working together through fast local area networks; in opposite to Grid, resources are not geographically spread. 27

Data Carousel system developed in STAR in order to coordinate requests for HPSS. 21

fair-share strategy to achieve equal resource usage among system users and groups with the respect of their priorities. 34

graph combinatorial structure that holds collection of vertices or 'nodes' and a collection of edges or 'links' that connect pairs of vertices. 41

Grid distributed and dynamic computer environment consisting of various loosely coupled resources acting together to perform large tasks. 26

heuristic experience-based technique used to speed up the process of finding a good enough solution, where an exhaustive search is impractical. 28

HPSS **H**igh **P**erformance **S**torage **S**ystem. software that manages petabytes of data on disk and robotic tape libraries. 15

job-shop optimization problem in which jobs are assigned to resources at particular times. 45

linear programming mathematical method for optimizing some objective over list of requirements represented as linear relationships. 57

load balancing methodology to distribute workload across multiple resources to achieve optimal utilization. 19

makespan total length of the schedule (that is, when all the tasks have finished processing). 66

NERSC/PDSF **N**ational **E**nergy **R**esearch **S**cientific **C**omputing Center / **P**arallel **D**istributed **S**ystems **F**acility at Lawrence Berkeley National Laboratory, USA. 18

NP-hard **n**on-deterministic **p**olynomial-time hard. Class of problems from computational complexity theory that are, informally, "at least as hard as the hardest problems in NP". 33

planning selection and organisation of actions in order to reach the goal or change of the system. 41

pruning eliminating branches of a search tree that do not contain (better) solution. 50

QOS **Q**uality **o**f **S**ervice. ability to guarantee a certain level of performance. 72

queue structure which stores tasks waiting for execution; tasks are selecting according to the applied dispatching rules. 28

race condition execution ordering of concurrent flows that results in undesired behavior. 32

RCF/BNL **R**HIC **C**omputing **F**acility at **B**rookhaven **N**ational **L**aboratory, NY, USA. 15

resource entity which executes, processes or supports the task (CPU, storage, link, etc.). 24, 43

scheduling allocation of resources to planned tasks over given time periods. 43

simulated annealing probabilistic metaheuristic for the optimization problem, where making a 'move' uses inspiration from annealing in metallurgy. 28

stream sequence of data packets used to transmit or receive information usually over the network. 46

symmetry breaking process of identifying and breaking symmetries (set of solutions which can be obtained by simple transformations from existing ones) in the search space. 51

thread smallest unit of processing that can be scheduled by an operating system. 48

unary resource resource with ability to execute only one task at any time. sometimes called also **serial**. 43

weighted graph graph with an associated label (weight) with every edge; often used in networking where weight represents speed/bandwidth of a link. 39

Chapter 1

Introduction and problem statement

A primary purpose of information technology and infrastructure is to enable people to perform their daily tasks more efficiently or flawlessly. Distributed computing offers large harvesting potential for computing power and brings other benefits as far as it is properly exploited [32]. On the other hand it introduces several pitfalls including concurrent access, synchronization, communications scalability as well as specific challenges such as answering key questions like “how to parallelize a task?” knowing where my data and CPU power are located. Unlike the resources addressed by a conventional operating system, these are distributed, heterogeneous and loosely coupled.

In data intensive experiments, like the one from High Energy and Nuclear Physics (HENP) community to which e.g. the STAR [1] [2] experiment belongs, the problem is even more significant since the task usually involves processing and/or manipulation of large datasets. The STAR experiment is primarily discussed and used for experiments and software deployment in this thesis.

For the colossal volumes of data being produced every year to be treatable, the technology and system must be manageable. In global collaboration, the needs for coordinated resource sharing and efficient plans solving the problem in a dynamic way are fundamental.

The massive data processing in a multi-collaboration environment often exploits geographically spread diverse facilities. Apparently, it will be hardly “fair” to users and hardly using network bandwidth efficiently unless we address and deal with planning and reasoning related to data movement and placement. This thesis addresses the paradigm of distributing the data focusing on one of the largest running physics experiment in the present. It exploits and applies the solving techniques for designing and building the

[1]Solenoidal Tracker at Relativistic Heavy Ion Collider is an experiment located at the Brookhaven National Laboratory (USA). See http://www.star.bnl.gov for more information.

automated planning and transferring system; enabling scientists to reach fruits of their research leveraging the potential of their resources.

1.1 Document structure and work overview

We will first outline some primary ideas of the content, thesis structure and underline the most important results that have been presented and published. We will arrange them along the timeline and point out also few other projects we have been involved in.

The first chapter introduces the environment of the physics experiment, background motivation and outlines the general scope of the project. Computing challenges and flow of data processing are highlighted while currently used data services are summarized. The *Tier* model leveraging regional resources is briefly described; and the experience of setting up a local computing site for offline analysis was published in [12].

The next chapter dives into the more detailed problem analysis related to data placement. We propose the concept of automated planning, introduce main use cases and requirements, including the goal proposal of the work and intermediate steps and milestones. Related work is described in section 2.2 and covers the difference and benefit of our planning approach with respect to the current attitudes. The problematic of fair-share and cache policy is covered in this part as well.

The principles of the constraint based modeling approach are covered in the third chapter and this chapter includes the underlying constraint based models. The initial ideas and simulated measurements were published in [69]. More elaborated planning heuristics were shown in [71] and in [72] we covered the full Constraint Programming model with computational complexity. In the sequel, further section 3.2 introduces the extension of the model and presents the Mixed Integer Programming approach. The mutual comparison and performance is immediately revealed in consequent text. The MIP approach was published in [70]. The last section of this chapter is devoted to "Holy Grail" in Data Grids - coupling computing elements with data movement inside the automated planner. We present the ultimate extension and modification to the model in order to cover full reasoning.

Chapter 4 offers the insider view into the technical implementation and software engineering part of our work, concentrating on the framework work-flow, database design and implementation details of the fundamental components. The main principles and real evaluation were published in [73]. The chapter includes practical proof of the principles, real-case experience and closes the loop of initially proposed requirements. The last chapter summarizes the overall status and outlines the eventual future direction.

The planning techniques we were researching in the network and data movement environment are applicable also in other fields. We investigated also the area of autonomous robots and related vehicle routing issues under constrained circumstances. The appendix is discussing this work which was published in [57] and recent results of the model based on finite state automaton can be found also in [5].

Along the international conferences the work has been continuously presented at STAR Collaboration Meetings and Regional Meetings as well. Several Czech&Slovak proceedings of local conferences were published too. There is definitely a lot of open topics for enhancement of the work and several issues need to be addressed in order to flawlessly deploy the system into an everyday production environment. The outlook is discussed in the final chapter of the thesis.

1.2 RHIC complex and STAR

The Relativistic Heavy Ion Collider [36] (RHIC) allows physicists from all around the world to study what the universe may have looked like in the first few moments after its creation. This may lead to better understanding why the physical world works the way it does, from the smallest subatomic particles, to the largest stars. The RHIC machine is located at Brookhaven National Laboratory (New York, USA), where many important discoveries were achieved (5 of them have been awarded a Nobel Prize).

It is a very versatile collider, capable of accelerating heavy ions (atoms having their outer cloud of electrons removed), primarily ions of gold, because its nucleus is densely packed with particles. RHIC collides two beams of ions head-on when they're traveling at nearly the speed of light. It provides access to the most fundamental building blocks of nature known so far - quarks and gluons. By colliding the nuclei of gold atoms together at nearly the speed of light, RHIC heats the matter in collision to more than a billion times the temperature of the sun. In so doing, scientists are able to study the fundamental properties of the basic building blocks of matter and learn how they behaved 15 to 20 billion years ago, when the universe was barely a split-second old. In parallel with this program, with increasing importance, there is also a unique proton-proton program to study proton spin structure [53].

The RHIC complex (Fig. 1.1) is composed of a "chain" of particle accelerators. Heavy ions begin their travels in the *Tandem Van de Graaff* accelerator, travel through a circular *Booster* where, with each pass, they are accelerated to higher energy. From the Booster, ions travel to the *Alternating Gradient Synchrotron*, which then injects the beams via another beam line into the two rings (identified as yellow and blue in Fig. 1.1) of

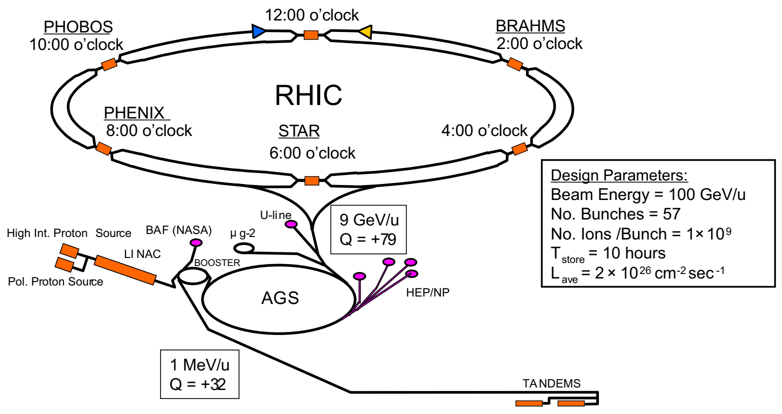


Figure 1.1: Layout of the RHIC complex at BNL - schematic drawing.

RHIC. Counter-rotating particle beams can cross at six intersections around the 2.4 mile RHIC ring. A different detector is located at each of the four intersection points currently in use. The RHIC was designed [27] to accelerate nuclei to top energy of 100 GeV/A for $A \gtrsim 200$ (nucleon's density), and at the same time being able to go continuously as low as 5 GeV/A with species from almost the full periodic table.

Just after the collision, thousands more particles form as the area cools off. Each of these particles provides a clue as to what occurred inside the collision zone. Physicists sift through those clues for interesting information.

Currently, there are four active experiments at the RHIC:

- **STAR** The Solenoidal Tracker at RHIC is used to search for signatures of the form of matter that RHIC was designed to create: the quark-gluon plasma.

- **PHENIX** The Pioneering High Energy Nuclear Interaction eXperiment, is a detector designed to investigate high energy collision of heavy ions and protons.

- **PHOBOS** & **BHRAMS** The PHOBOS detector was designed to examine and analyze a very large number of unselected gold ion collisions. The BHRAMS was designed to measure charged hadrons over a wide range of rapidity and transverse momentum to study the reaction mechanisms of the relativistic heavy ion reactions at RHIC and the properties of the highly excited nuclear matter formed in these reactions.

This work was supported and primarily intended for the STAR experiment, within which the research was carried out.

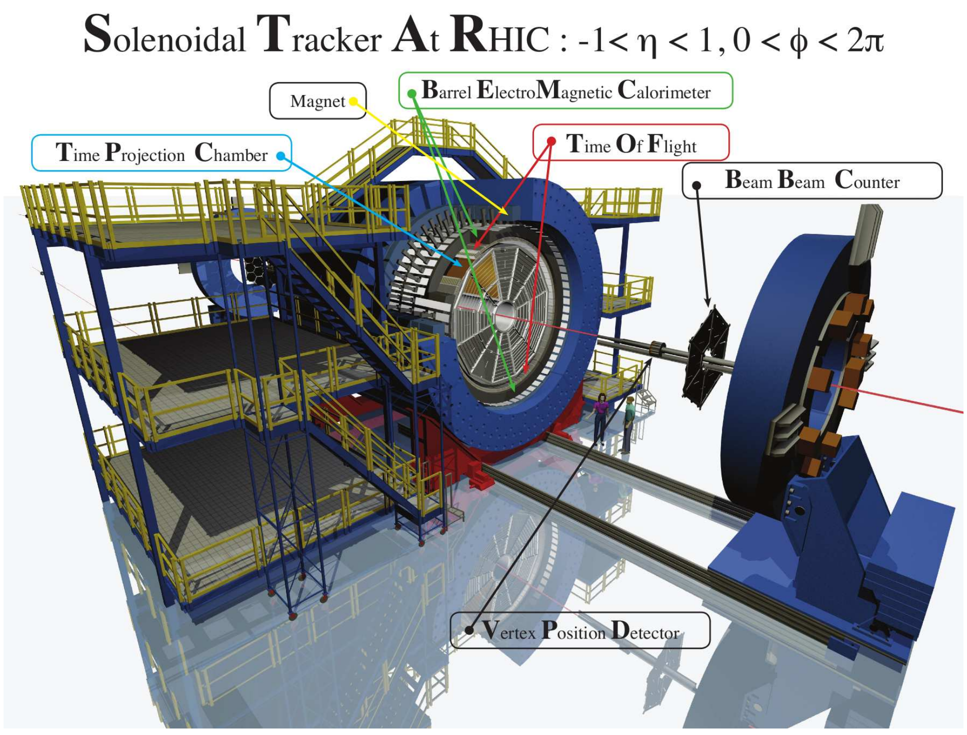

Solenoidal Tracker At RHIC : -1< η < 1, 0 < φ < 2π
Magnet
Barrel ElectroMagnetic Calorimeter
Time Projection Chamber
Time Of Flight
Beam Beam Counter
Vertex Position Detector

Figure 1.2: Perspective view of the STAR detector, with a cutaway for viewing inner detector systems.

1.2.1 The STAR experiment

STAR is a general-purpose high energy physics detector (see Fig. 1.2) with a variety of subsystems optimized for the detection of diverse types of particles emitted from the collisions of heavy ions or polarized protons. It is featuring detector systems for high precision tracking, momentum analysis, and particle identification at the center of mass rapidity. The large acceptance of STAR makes it particularly well suited for event-by-event characterizations of heavy ion collisions and for the detection of jets. With relatively short run periods, high statistics data can be taken that will allow analysis of unprecedented detail over the energy range planned.

STAR is situated in the six o'clock position in the RHIC ring. The experiment is a large collaboration of more than 500 scientists and engineers representing $\approx$ 60 institutions in dozen countries. The geographical distribution of the institutions in the STAR Collaboration is given in Table 1.1.

STAR collects several gigabytes of raw data every second during data taking. These raw data are promptly compressed and translated into a format that can be analyzed by physicists, but even the produced dataset contains millions of interesting "events"

Country	Institutions	Percentage
USA / North America	24	46%
Europe	12	23%
Asia (China / Korea)	8	15%
India	6	12%
South America	2	4%

Table 1.1: Geographical distribution of institutions in the STAR Collaboration as of 2008.

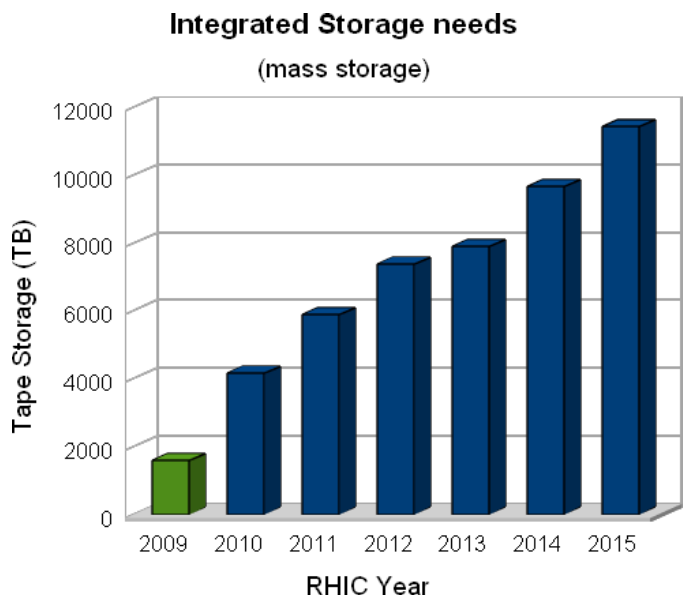


Figure 1.3: Projection of data for the STAR experiment.

and measures in the hundreds of terabytes up to several peta bytes for a given year of experimental running.

1.3 Computing challenges

Very often, the physics topics of HENP experiments are statistically driven. In order to get a significant statistical data sample, the experiments have to generate and acquire enormous data for further analysis. During the normal operation mode, the STAR detector produces raw data with the speed of $\approx 500-600$ MB/s (an average size of an "event" is 0.62 MB and the frequency of acquisition system is 1000 Hz). The Figure 1.3 is outlining the continuous growth in STAR's stored data at tape drives, reaching the magnitude of several Peta bytes.

The data volumes will even more likely grow in the future generations of such experiments. Running an analysis means, firstly, to develop an application (usually using a data analysis framework), then obtain input data the application depends on, and, finally, execute these tasks on computer elements and produce user derived results. From the yearly data sets, the experiment may produce many physics ready derived data sets which differ in accuracy as the problem is better understood as time passes. In addition to a typical Peta-scale challenge and large computational needs, such running experiments acquiring a new set of valuable real data every year need to provide data for physicists from previous years and consequently at any point in time.

With such demands from hundreds of collaborators in parallel, requesting time and computing wise intensive processing of large-scale data sets, one can easily see the eminent needs for efficient storage and computing solution.

1.3.1 Flow and data management in STAR

We will outline the basic procedure the STAR experiment has developed for handling and managing acquired data (Fig. 1.4). The raw detector data is immediately transferred to the set of low level tuned Linux boxes called *Buffer box*. Using the water marking, if the use of local disk of any buffer box machine exceeds some level, raw files are moved (using *pftp* protocol) to the tape storage at local computing facility at BNL, called RHIC Computing Facility (RCF) [22], using the optical fibers. The Mass Storage System (MSS) at RCF/BNL is called High Performance Storage System [2] (HPSS) and is based on the robotic tape system. The graph from Fig. 1.5 shows the data mover statistics from STAR online to Mass Storage. This system works as a tertiary storage repository for STAR where all raw data sets reside.

The raw data needs to be processed by the reconstruction software with appropriate calibration information to be usable for later physics analysis. This raw reconstruction process (main operation is tracking the particles) happens on the computing farm nodes. Reconstruction jobs are submitted to the machines using local Resource Management System (on top of the Condor dispatcher) and jobs pull data from HPSS to local disks for full chain processing.

In addition, to relieve the RCF resources, part of the raw data (15%) are planned to be processed in Korea Institute of Science and Technology Information (KISTI) in Asia. Recent promising transfer studies and tests showed the inter-continental link full bandwidth saturation is possible in a sustainable mode.

The processed files are called Data Summary Tapes (DSTs), often reffered to as DAQ

[2]HPSS: http://www.hpss-collaboration.org/

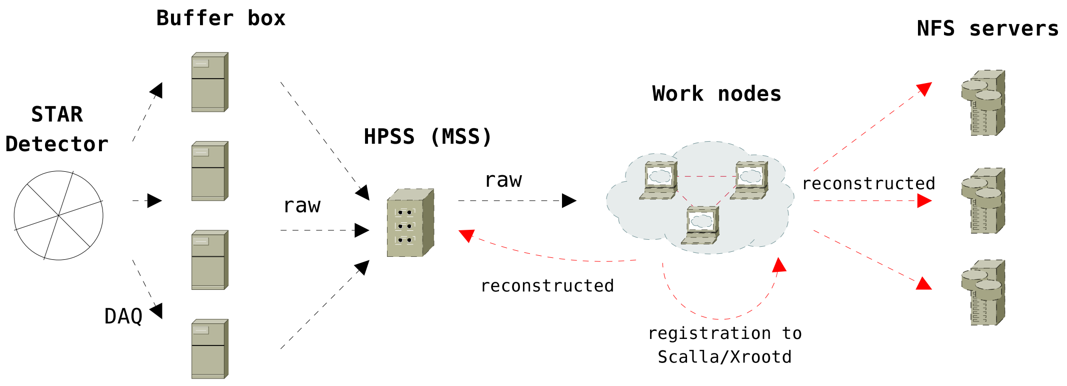


Figure 1.4: Schematic drawing of data flow in STAR. The data flow starts from the single point - the detector.

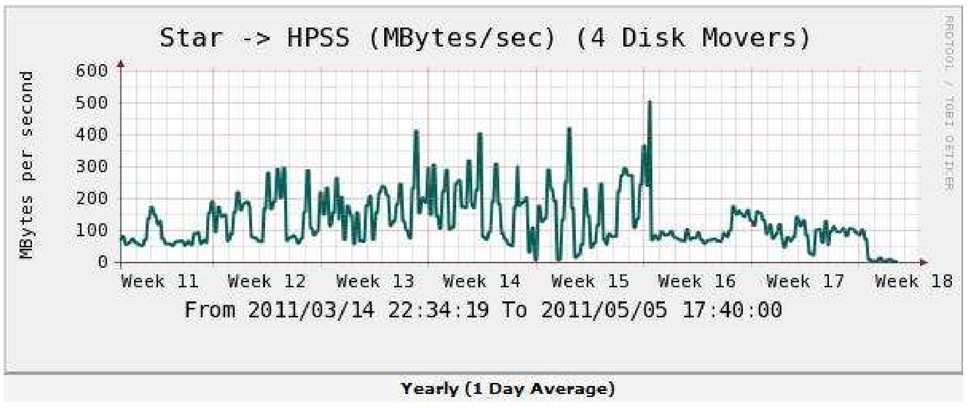


Figure 1.5: Data mover statistics from STAR online to Mass Storage. Over the run period, the averages sustained at the level of 250 MB/sec.

	DAQ raw		**DAQ reco**		**DAQ μDST**	
Run year	*avg size*	*count*	*avg size*	*count*	*avg size*	*count*
2010	1.143 GB	774 808	1.340 GB	25 782	208.720 MB	240 320
2009	925.836 MB	250 102	2.447 GB	62 111	333.256 MB	255 065
2008	436.765 MB	353 947	795.685 MB	66 255	136.375 MB	489 013
2007	470.104 MB	317 167	564.371 MB	325 511	165.575 MB	331 062
2006	433.809 MB	115 472	583.802 MB	135 339	84.915 MB	135 365
2005	441.356 MB	326 809	323.758 MB	528 850	71.419 MB	545 502
2004	461.349 MB	406 091	364.974 MB	399 003	153.526 MB	453 405
2003	479.092 MB	124 956	95.428 MB	209 552	26.339 MB	206 428

Table 1.2: Average file size of different file types during the years 2003 - 2010.

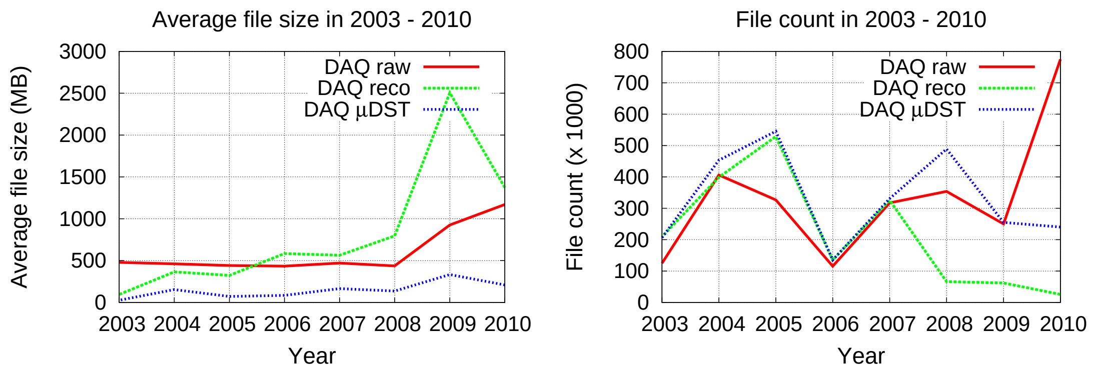


Figure 1.6: **Left.** Average size of different file types. **Right.** Counts of different file types. The raw data sets from 2010 have not been fully processed at the time of writing this text.

reco, and contain all the information the physicists can use later. Mostly, physicists do not need all the saved information in DST files and since the files can be fairly big, for the space and later computing efficiency, the files are transformed into the smaller files, with only the essential information left. These files are called μDST. All copies of the raw, DST, and μDST files are copied and kept back on the HPSS. One has to also consider the fact, that STAR has several productions of μDST files from the single set of raw files, passed with different reconstruction criteria and software settings.

The Table 1.2 and the graphs from Figure 1.6 represent the average size of different file types together with their appropriate count as a function of years. With the installation of the DAQ1000 upgrade in 2009, the size of the raw datasets almost doubled. We can see that accumulated size of reconstructed files (over all passes) usually oversteps the size of raw data sets.

	2010	**2011**	**2012**	**2013**	**2014**	**2015**	**2016**
Typical number of Tier-2	4	3	4	5	4	4	3
Bandwidth @ Tier-2	0.84	1.06	1.08	1.24	1.67	1.47	1.47
Bandwidth from BNL	2.24	2.11	2.87	4.14	4.44	3.93	2.94
Bandwidth from LBNL	1.12	1.06	1.43	2.07	2.22	1.96	1.47

Table 1.3: Data transfer rates for sustaining redistribution of μDST to other *Tier-2* centers.

Tier model

Given the international composition of the collaboration, the STAR Software and Computing (S&C) model has naturally evolved toward a Tier structure, similar to that utilized by other major international S&C efforts. The *Tiers* (*Tier 0, 1, 2*) are defined by the services and capabilities available at the institutions within a given classification.

Since BNL is hosting the STAR detector, it is the unique *Tier-0* center by definition. A *Tier-1* center in STAR is defined as a site providing persistent storage, Mass Storage System (MSS), as a local service and also a site providing resources (storage or processing power) to level of 15% or more to perform any specific task. Since 2000, the **P**arallel **D**istributed **S**ystems **F**acility at the **N**ational **E**nergy **R**esearch **S**upercomputing **C**enter (NERSC/PDSF) at Lawrence Berkeley National Laboratory has been the only *Tier-1* center in STAR. A STAR *Tier-2* site is an institution having several tens of TB of storage which can be utilized for local or regional needs for user analysis. Wayne State University, MIT, and NPI/ASCR in Prague are examples of STAR *Tier-2* centers. The distribution of "physics ready" data allows for an enhanced productivity where they become available. Table 1.3 presents typical number of *Tier-2* sites for upcoming years (line 1) and expected network estimate. From the STAR observations typical *Tier-2* site replaces the local data on the order of 4 times a year. The projected numbers for total bandwidth required (lines 3 and 4) assume a target goal where $2/3$ of institutions would acquire data from BNL and $1/3$ from LBNL.

Magellan Cloud STAR has recently made also massive use of Magellan Cloud facility where two government-funded testbeds NERSC/PDSF and the Leadership Computing Facility (ACLF) at Argonne National Laboratory (ANL) joint virtual clusters made up of IBM iDataplex solution in order to provide a cost-effective and energy-efficient paradigm for science. This collaboration resulted in a real-time cloud-based data processing system running a coherent cluster of over 100 Virtual Machines from three Magellan resource pools - *Eucalyptus* [3] at NERSC, *Nimbus* [4] at ANL and *OpenStack* [5] at ANL. The total number of cores has exceeded 800. While set at national laboratories, Magellan cloud has predictable network path (in opposite to true commercial cloud).

Data services The RCF facility at BNL has three major components, namely HPSS, the Linux Farm and the centralized disk system. Various storage systems mutually differ; and usually huge and cheap amount of storage is payed off by high latency time or data access inconvenience. It is clear that application benefits often require the data to be "close enough". By this we mean the data access from the application viewpoint must be smooth, prompt and reliable. Therefore, the data movement from MSS to "closer" storage elements, data placement strategy, and the data access solutions are the key blocks in STAR S&C program.

The RCF has provided central storage (based on BlueArc or similar solution) made available to the computational nodes via Network File System (NFS). As it was explained above in the text about data flow, the reconstructed μDST files are moved back to the permanent HPSS storage, however the system populates these files also to other data services. Daemons called *Spiders* constantly monitor the presence of reconstructed files on NFS and also on distributed storage system (explained in the following section 1.3.2). This infrastructure, keeping a catalogue of files up to date, also allows for further data management tasks such as dataset replication on other elements, detecting missing files in HPSS and automatic deletion of files from NFS (keeping a copy on distributed storage). The distributed storage is primarily used for data access from jobs.

STAR has been intensively involved in deploying distributed model for storing and accessing the data. The main reasons behind focusing on the distributed solution and its basic advantages are:

- **Scalability:** distributed system can better scale with increasing requirements for the capacity
- **Load balancing:** the distribution of data implies spreading the load among the elements
- **Fault tolerance:** avoiding the centralized single point of failure leads to improved fault tolerance

[3]Eucalyptus: http://www.eucalyptus.com
[4]Nimbus: http://www.nimbusproject.org/
[5]OpenStack: http://www.openstack.org

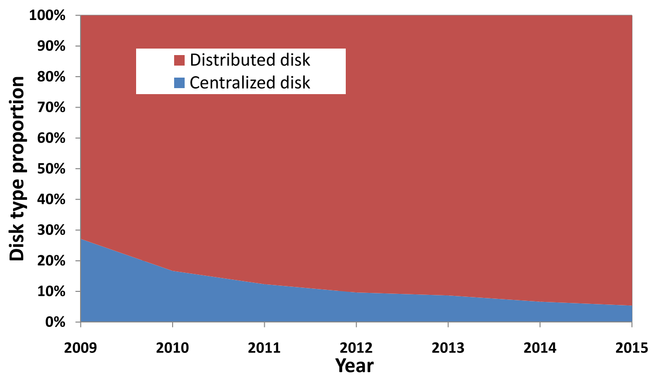


Figure 1.7: Confrontation of centralized and distributed capacity of storage in STAR.

The larger portion of the overall storage space used to be served by centralized systems in the past; however, as we can see from the graph in Fig. 1.7 most of the data resides on the distributed disk space and this tendency will be even more dominant in the future. Since 2006, STAR has been using Scalla/Xrootd system to aggregate and access data in a scalable way. We will outline the main concept of this approach.

1.3.2 Scalla system

The Scalla (Structured Cluster Architecture for Low Latency Access) [28] software suite offers a framework for aggregating a commodity hardware to construct large fault-tolerant high performance data access. The suite consists of a file server called *xrootd* and a clustering server called *cmsd*. The xrootd server was developed for the ROOT [6] analysis framework to serve root files. However, the server is agnostic to the file type and provides byte level access to any type of file. The cmsd server is designed to provide file location functionality and cluster health and performance monitoring.

One of the fundamental principles of the suite is its structured hierarchical subscription model. That is, cmsd's connect to other cmsd's in order to form a compact $B-64$ tree, as shown in Figure 1.8. A special cmsd, called the redirector, sits at the root of the tree. This server is given the role of a manager. The manager is responsible for issuing file queries and collecting the responses from nodes lower in the tree. A server cmsd is in one-to-one correspondence with a data server (i.e., a machine that serves data files). This kind of architecture scales very quickly with a minimum amount of message traffic (due to the logarithmic height of a search tree). The limit of 64 nodes is deliberate. Sixty-four allows efficient bit-slice vector operations using 64-bit integers and deterministically lim-

[6]ROOT - A Data Analysis Framework: http://root.cern.ch

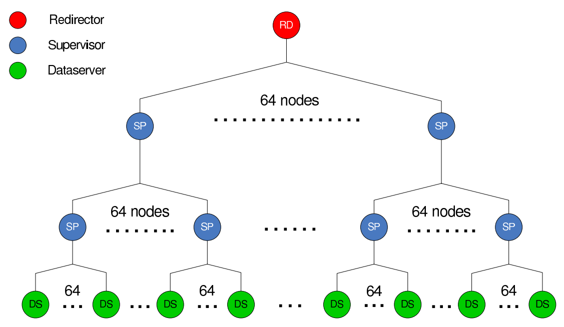


Figure 1.8: Example of B-64 tree structure used for clustering servers (and aggregating the distributed storage).

its the amount of work any particular server needs to do. The latter is critical in providing a deterministic time bound for file queries.

DataCarousel

In the STAR environment, files not available at any xrootd data server are transferred from the MSS as shown in Fig. 1.9. The component called *Data Carousel* was developed for this purpose - to coordinate the requests which would otherwise be initiated from all data servers within the Scalla/Xrootd architecture and other client requests as well (this would lead to a collapse of MSS or inefficiency). The DataCarousel works simply as a HPSS front-end, managing the asynchronous requests. It aggregates the set of requests from clients, re-orders them according to the internal policy, and passes them further to the HPSS. The system itself is composed of a light client program, a plug-and-play policy based server architecture component and a permanent process interfacing with the mass storage using HPSS API calls. The client and server interacts via a database component isolating client and server completely from each other. Policies may throttle the amount of data by group (quota, bandwidth percentage per user) but also perform tape access optimization such as grouping requests by the tape identification number.

Recently, the Scalla/Xrootd system development is trying to exploit also the benefits of faster and lower latency Wide Area Networks (WAN). The principal idea is that missing files can be retrieved over WAN from other Scalla cluster (geographically in a different location) on the fly. This operation allows to start an application at the cluster even if some files are not available. The metric used for making decision where to redirect a client considers only the load of the servers. As an addition, the site hosting the Scalla cluster has to allow direct incoming connections from remote clients, what is often not always feasible in such organizations. There is definitely a need for utilizing

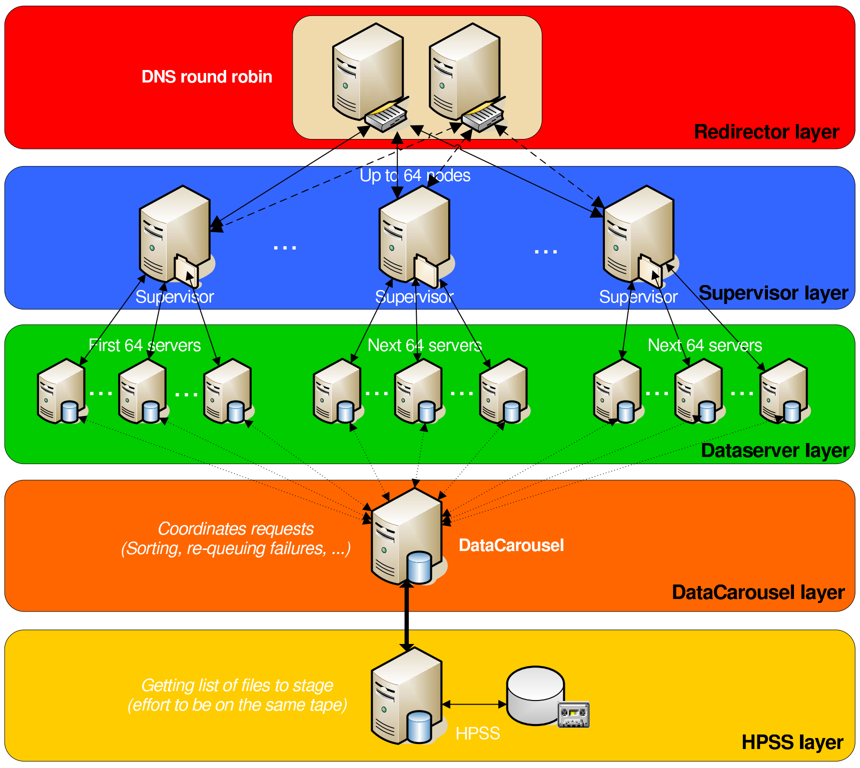


Figure 1.9: The Scalla/Xrootd overview with DataCarousel integration as deployed in STAR.

the more and more available sources over the WAN and the rising demand for a “decision maker”. The question whether to retrieve missing files on fly, or whether to distribute the data in advance to achieve the proper optimization is then just leveraging the decision in one way or another.

Chapter 2

Problem analysis

This chapter, after the preliminaries, introduction and overview of the STAR's computing challenges, is devoted to more elaborated problem analysis. We will introduce the goal of the automated planning system remarking the main workflow principles, optimization features and the motivation for further software architecture. We will discuss the related works, give summary of the current approaches and techniques, and bring the basic solving attitudes the system is based on. Because of the broad range of factors the components have to deal with, we will not step into the deep discussion about all of them, but present an overview of some like fair-share, cache policy, etc.

2.1 Use case and requirements

The purpose of our research and work is to design and develop an automated planning system acting in a multi-user and multi-service environment as shown in Fig. 2.1. The system acts as a "centralized" decision making component with the emphasis on **optimization**, **coordination** and **load-balancing**. The optimization guarantees the resources are not wasted and could be shared and re-used among users and sources. Coordination ensures multiple resources do not act independently so starvation or clogging do not occur, while load-balancing avoids creating bottle-necks on the resources. The intent is not to create another point-to-point data transfer tool, but to use available and practical ones in an efficient manner.

We describe the most important optimization characteristic with the help of figures Fig. 2.2 and 2.3. Let us suppose there are requests for the same (or overlapping) dataset from two users, while each of them needs the dataset to be processed at his/her specific location. The system has to reason about the possible repositories for the dataset, select the proper ones for every file (the granularity is specified by the files in our case) and pro-

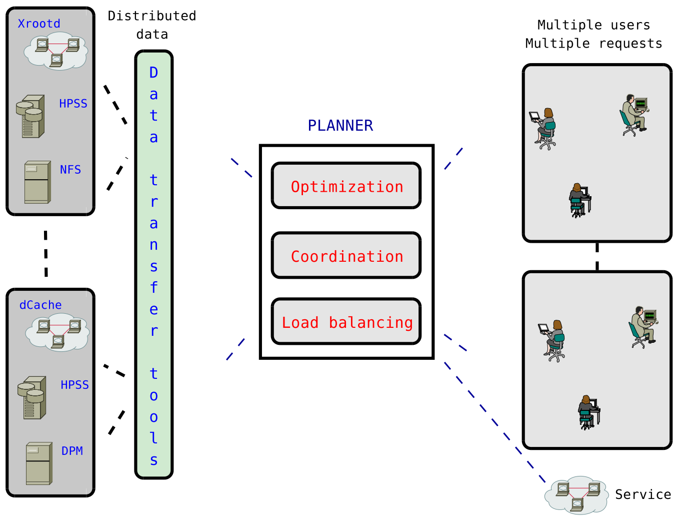


Figure 2.1: General view of the automated planning system. The goal is to achieve controlled and efficient utilization of the network and data services with a proper use of existing point-to-point transfer tools. At the highest level of abstraction, the planner should appear as a “box” between the user’s requests and the resources.

duce the transfer paths for each file. The output plan should be optimal with an objective of the overall completion time of all transfers. Thus, this optimization characteristic is focusing on the network structure and respective link bandwidth. As illustrated in Figure 2.2, it is conceivable in our example that optimization will cause data movement to occur once on some network links while datasets will be moved to two different destinations.

Moreover, the files are usually served by several data services (such as Xrootd, Posix file systems, Tape systems [58]) with different performance and latencies. Therefore, the optimization and reasoning on where to take the files available from multiple sources

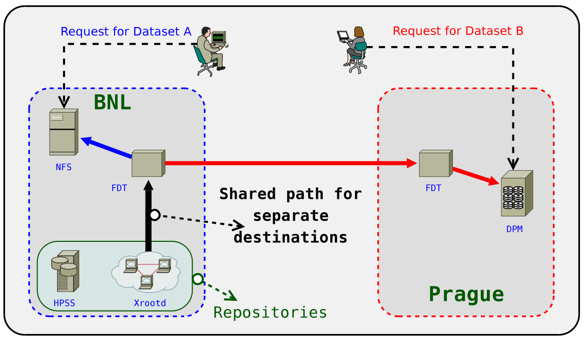


Figure 2.2: Optimization of the transfer paths with regards to the network structure and link bandwidth. Some network path may be re-used to satisfy multiple requests for the same data.

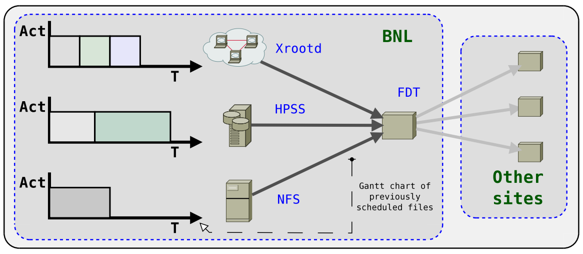


Figure 2.3: Optimization of the transfer paths with regards to the different data service performance/latency. Multiple sources for the same data may be naturally combined alternatively to avoid overload and service clogging.

choice will allow making the proper selection for a file repository, respecting their intrinsic characteristic (communication and transfer speed) and scalability (Fig. 2.3). In other words, as soon as multiple services and sources are available, load balancing would immediately be taken into account by our planner.

2.2 Related works

The needs of large-scale data intensive projects arising out of several fields such as bioinformatics (BIRN, BLAST), astronomy (SDSS) or HENP communities (STAR, ALICE) have been the brainteasers for computer scientists for years. Whilst the cost of storage space rapidly decreases and computational power allows scientists to analyze more and more acquired data, appetite for efficiency in Data Grids becomes even more of a prominent need.

While the “traditional” *Computational Grids* were developed due to the need to solve problems that require processing a large quantity of operations, the *Data Grids* [13] purpose was extended for another dimension. They primarily deal with data repositories, sharing access and management of large amounts of distributed data. In such systems many types of algorithm, such as replication, are important to increase the performance together with data copy and transfer. In other terms, data location is important in such a type of scheduling, because it is not only important to allocate tasks, jobs or application to the fastest and reliable nodes but also minimize data movement and ensure fast access to data.

Decoupling of job scheduling from data movement was studied by Ranganathan and Foster in [47]. Authors discussed combinations of replication strategies and scheduling algorithms, but not considering the performance of the network. The nature of high-energy physics experiments, where data are centrally acquired, implies that replication to geographically spread sites is required in order to process data distributively. Intention to access large-scale data remotely over wide-area network has turned out to be highly ineffective and a cause of often sorely traceable troubles.

The authors of [55] proposed and implemented improvements to the Condor, a popular cluster-based distributed computing system. The presented data management architecture is based on exploiting the workflow and utilizing data dependencies between jobs through study of related direct acyclic graphs. Since the workflow in high-energy data analysis is typically simple and embarrassingly parallel without dependencies between jobs these techniques don't lead to a fundamental optimization in this field.

Sato et al. in [54] and authors of [46] tackled the question of replica placement strategies via mathematical constraints modeling an optimization problem in Grid environment. Solving approach in [54] is based on integer linear programming while [46] uses Lagrangian relaxation method [21]. The limitation of both models is a characterization of data transfers which neglects possible transfer paths and fetching data from a site in parallel via multiple links possibly leading to the better network utilization.

We focus on this missing component considering wide-area network data transfers pursuing more efficient data movement. An initial idea of our presented model originates from Simonis [56] and the proposed constraints for traffic placement problem were expanded primarily on links throughputs and consequently on follow-up transfer allocations in time. The solving approach is based on Constraint Programming technique, used in artificial intelligence and operations research. One of the immense advantages of the constrained based approach is a gentle augmentation of the model with additional real-life rules. Constraints identify the impossible and reduce the realm of possibilities to effectively focus on the possible, allowing for a natural declarative formulation of what must be satisfied, without expressing how.

2.2.1 Static vs. dynamic scheduling

A workflow application is typically a set of tasks linked by producer-consumer communications. In general, it can be represented as a direct acyclic graph (DAG), where the node is the individual job and edge represents the inter-job dependence. If we restrict ourselves to Grid environment and data transfers, a nodes can represent data transfers over individual links and the connection between nodes represents the order of transfers. One of the key functions of a workflow management systems [9] is to schedule and manage the tasks on shared resources to achieve high performance. When it comes to implementation, the planner and executor are two core components in terms of how the resource mapping decision is made and how a task (transfer) is planned. However, the characteristics of the Data Grid environment make the coordination of its execution very complex [11, 68].

Static approach, i.e. classical planning/scheduling, is built on creating full plan ahead, where the planner makes the global decisions in favor of the entire workflow performance. Assuming the environment is static, there is no uncertainty in the behavior of resources and activities which are given in advance and assuming the performance of the planning system is robust enough to cover the scale of the problem, this approach can lead to (near) optimal plans.

However, in a grid environment static strategies may perform poorly because of the Grid dynamics: resource can join and leave at any time; individual resource capability varies over time because of internal or external factors; and requests (tasks) arrive as time goes. Also computation cost of each job is not easy to accurately estimate, which is the foundation of any static approach [64].

Because of several performance requirements and optimization criteria, the Grid problem is multi-objective in its general formulation. Therefore isolated simple heuristics (also known as dispatching rules and policies) like First-Come First-Served, Earliest Deadline First, Shortest Job First, etc. cannot meet all the needs of the complex objective. However, since they are optimal for specific problems, easy to implement, and adapt well to high dynamics, they (or some combination of them) are often used in theory and practice [49]. To mention some production scheduling systems based on queue-based policies, one can look at Condor [59], LSF [66] or PBS [20].

Local search (LS) [29], as a metaheuristic method for solving computationally hard optimization problem, has been also applied to several Grid scheduling domains. Several LS methods based on Hill Climbing have been studied in Ritchie an Levine [51]. Simulated annealing, which accepts also worse solutions with certain probability, has been proposed for Grid scheduling by Abraham et al. [1] and Yarkhan and Dongarra [67]. Tabu search is more sophisticated and usually requires more computation time to reach good solutions. Several use were reported [33, 65] with a pursuit to optimise an initial schedule computed with the help of dispatching rules in a dynamic Grid environment. LS methods are often considered to be very time consuming, but authors showed that with an incremental approach based on a previously computed schedule one can outperform queue-based approaches with very reasonable runtime requirements.

Another large family methods for solving combinatorial optimization problems is population based heuristics. They often require large running time, but when objective is reduced to find feasible solutions of good quality the use of them may be appropriate.

Genetic algorithms for Grid scheduling have been addressed by Abraham et al. [1], Braun et al. [10], Zomaya et al. [75], Page and Naughton [43].

There are many other approaches that have been applied to particular Grid related problems and research papers frequently report some benefits of one sort or another. However, most of the work has been done in academic ground within some simulated environment and to our best knowledge most of them have not been applied and deployed for the real production environment facing a comparable scale of problems like STAR experiment does.

Designing and implementing the automated planning system in a dynamic environment like Data Grid is, will not be fruitful unless we address the three main issues: *(1) Accuracy of estimation*. Estimating the computation cost of a job (w.r.t. either computing or transfer) is the key success factor, but at the same time the system can be hardly effective if it has to reason about all peculiarities from the environment. The proper abstraction of the real world is needed to provide balance between accuracy and complexity. *(2) Adaptation to dynamic environment*. Pure static approaches assume that resource and task set is given and fixed over time. Since this assumption is not always valid, the adaptation to the changing condition is needed. In other words, the system has to provide proper balance between being **deliberative** and **reactive** at the same time. *(3) Separation of planner from executor*. Fundamentally the first two issues are related to the lack of collaboration between planner and executor. Without a cooperation the planner cannot be aware of the grid environment change and cannot adapt to the more accurate estimations.

In the following chapters we will deep into the process of designing, implementing and deploying the automated planning system in data intensive evironment targeting the above mentioned issues.

2.3 Workflow analysis

In this section we explain the workflow of the planning process and further file transfer execution with regard to the storage issue. The system offers file transfers over intermediate sites if it leads to higher performance or load balancing. By higher performance in this case we mean achieving a higher network throughput (combination of faster network links), while load balancing assures the network load is spread through the diverse path if it is possible and beneficial. Since the transfer paths can include the intermediate site, one can clearly see there is a need for a temporary storage at such a site. To handle this there are two possible solving approaches.

The first one addresses the storage component directly in the model. In this case, the model is extended for additional restrictions (namely the cumulative resources) which emulate the selected disk usage increase (during the transfer) or decrease (after the transfer). While the model would appear as "fully consistent" and encompassing all the details, this approach brings further complications to the model leading to the higher solving complexity and also the fact that real data transfers do not always correspond to the computations from the model (network is a dynamic environment with a lot of unpredicted behavior). Therefore, handling these "exceptions" would be necessary outside of the planner anyhow.

The second approach is based on the idea that the model itself should stay simple and the storage resource handling is achieved dynamically during file transfers. To explain how we handle the restrictions from a storage space without involving it into the planner, let us first look into the workflow of the transfer mechanism over the link (Fig. 2.4). We start with the general (high-level) explanation and later in the text we address several details.

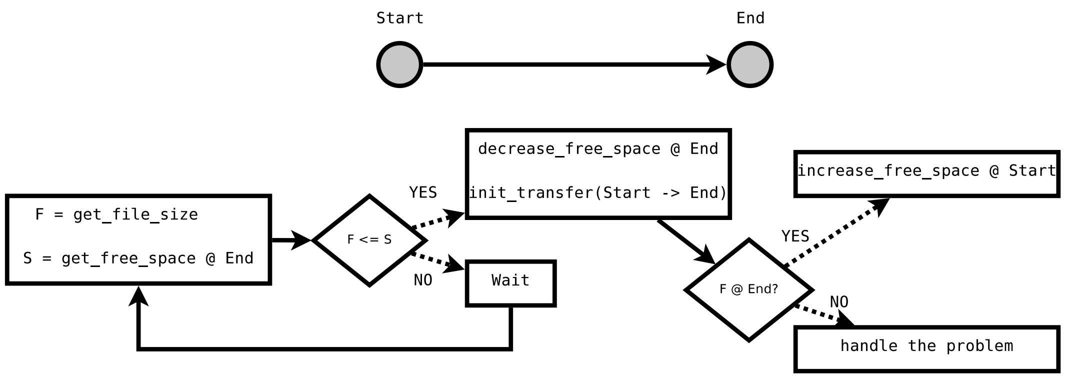


Figure 2.4: Flowchart of the transfer mechanism.

Before the transfer tool is executed, the free space at the destination site is checked and compared with the required space for temporary caching the file. If there is enough space, the actual information about the available space is updated and the transfer starts. Otherwise, the system waits until some other process relieves the space. Upon the successful transfer, the space at the starting site is released (if not, the transfer problem is handled). Generally, each consequent transfer atomically checks and updates the required space for itself.

When the next iteration of the planner is called, several files from some previous plan may be still in a transfer. They may already be in a transfer on the last portion of the planned path or still waiting to be transferred from an intermediate site. The system keeps track of the current status of all files and the planner considers this information.

Hence, if some link is being occupied (or files are waiting to be transferred over it), the planner includes this "waiting" time into the reasoning. When some link is creating a bottleneck because of the low bandwidth or similarly some site due to the very limited storage space, the planner will automatically reuse other resources if it will lead to the better (shorter) plan. Therefore, the dynamic storage handling outside of the planner will not cause the separation between the plans and the reality.

This process is illustrated in figure 2.5. For the simplicity, let the network consist of only three sites. In the beginning, there are no files in a transfer, so no link is occupied. The planner produces the transfer plan for requested files *A*, *B*, and *C* , for demonstration let us assume all files are available at the site *Start*, have to be transferred to the site *End* through the intermediate site *Inter*. After a while, files *A* and *B* are already transferred at site *End*, and file *C* is in a transfer from *Start* to *Inter* site. When the next pass of the planner is executed, the two links will be occupied, therefore the next requested file *D* will be transferred directly from *Start* to the *End*.

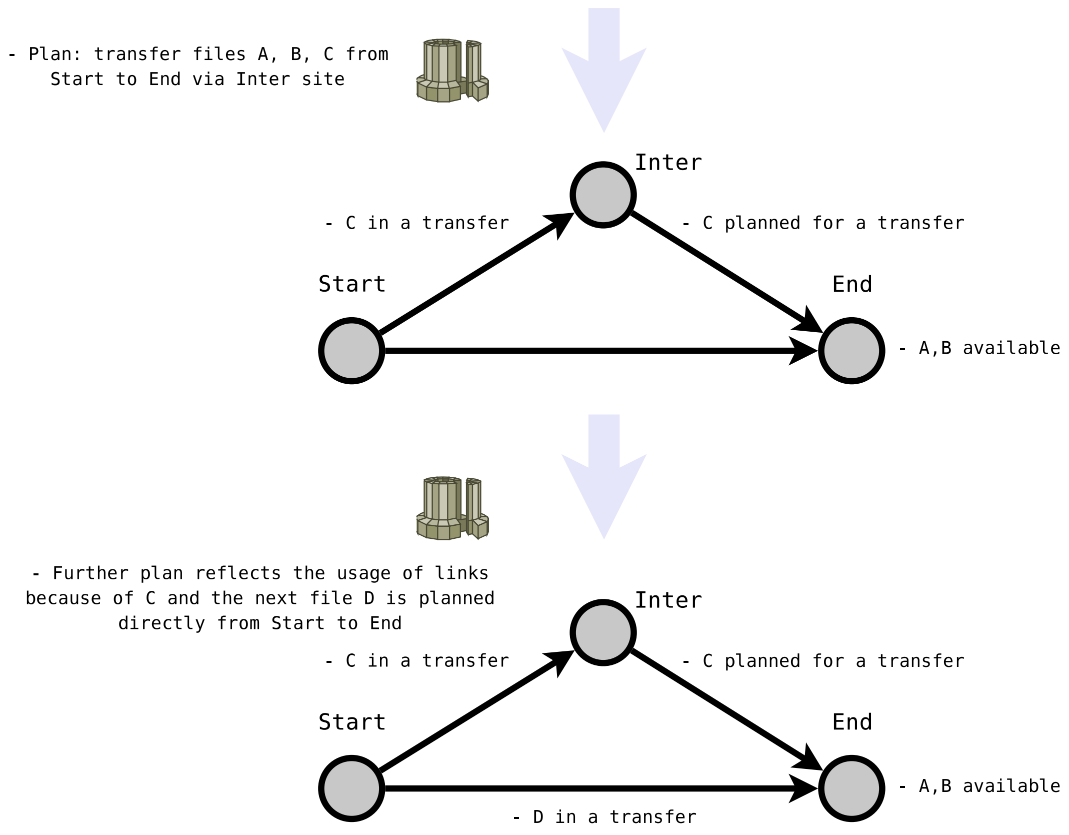

- Plan: transfer files A, B, C from Start to End via Inter site
Inter
- C in a transfer
- C planned for a transfer
Start
End
- A,B available
- Further plan reflects the usage of links because of C and the next file D is planned directly from Start to End
Inter
- C in a transfer
- C planned for a transfer
Start
End
- A,B available
- D in a transfer

Figure 2.5: Illustration of links usage by the planner in time.

Regarding the storage issue, an important observation we need to mention is that the time while the file is stored at intermediate site is assumed to be short and the total space needed at the site not to be a critical part of the mechanism (typical available disk space versus an average space taken by files in a current transfer). In other words, if we look at the flowchart in Fig. 2.4, the check whether there is enough space for a file at the intermediate site would be mostly positive. However, from the flowchart one can also see a possible *race condition [62]* which can theoretically occur. A race condition is an execution ordering of concurrent flows that results in undesired behavior. Since the transfer tool, in case of not enough storage space at the destination site, relies on another tool instance to release the space, there exists a possibility of a deadlock (as can be seen in Fig. 2.6).

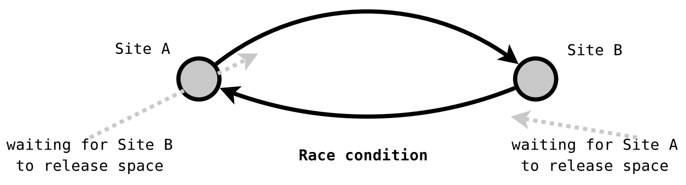

Site A
Site B
waiting for Site B to release space
Race condition
waiting for Site A to release space

Figure 2.6: Schematic drawing of a theoretically possible race condition.

Closer look at the flowchart 2.4 raises a question when to consider a transfer instance as not possible to act due to the locked destination storage. Any “try-wait” block can lead to the resource starvation and there is a need for maximum number of tries (with smartly defined intervals in between). After reaching the maximum tries and if the storage lock still persists, the deadlock state has to be checked and solved.

To be able to detect and jump out of a deadlock, we propose to use a directed cycle detection in a resource graph. During operation, if the transfer tools are waiting and we detect an oriented cycle in the corresponding graph they form, the race condition occurs. To solve it we can restart one of the transfers from the beginning, while the other one resumes itself. A transfer path (partial task of the plan) is selected, the intermediate file copies are deleted and the transfer is marked to be planned again in the next batch. The space is released and other transfer can be resumed.

Similarly, flowchart 2.4 also uncovers the possible transfer problem due to any reason. The point-to-point transfer can fail because of a broken link (which can be temporal or permanent), software failure (client or server side issues), or other third parties containment. Analogous to the previous case, there is a need for reasonable number of retries with updating and storing the failure information, up to the point when the link is marked as broken. This indicates to the next iterations of the planner to exclude the link from the reasoning until it will get back to the normal operation (see Fig. 2.7).

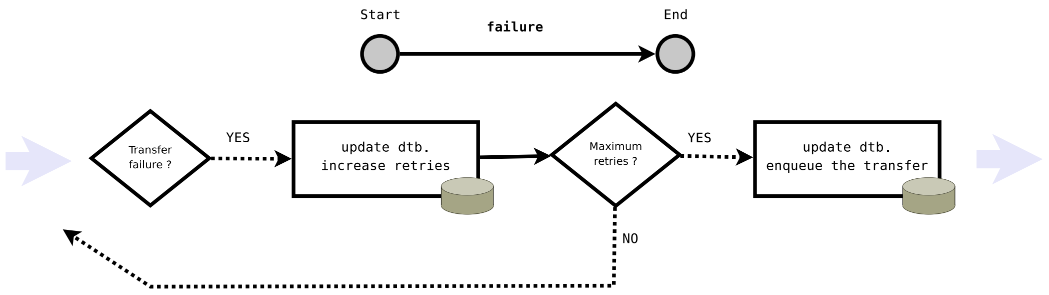

Start
failure
End
Transfer failure ?
YES
update dtb. increase retries
Maximum retries ?
YES
update dtb. enqueue the transfer
NO

Figure 2.7: Handling the possible failure of a transfer over the link.

2.3.1 Working with chunks

We decided to work and plan with chunks (batches) of files instead of reasoning about the full waiting list at once. This brings a lot of benefits and simplifications:

- Adaptability to the changing network situation.
- Smaller the input faster the computation we get. This is especially important, since the problem is NP-hard (as we will see in 3.1.3).
- Adaptability to the new incoming requests from different users.
- User fair-share can be thus realized using queue-based mechanism.

The global optimality does not have to be achieved when considering chunks instead of the whole list of queued files, but the very first measurements showed that planning file transfers by small chunks (tested different sizes) does not lead to significant impact on the lost on global optimum. This is demonstrated in the following graphs (Fig. 2.8).

The *X* axes denote the number of files in a request while *Y* is the time (in units) needed to generate the schedule and percentage loss on optimal solution. We can see that time to find an optimal schedule without any additions grows exponentially and is usable only for a limited number of files. Splitting the input into chunks results both in the very promising running time and also in the quality of the makespan.

2.4 Fair-share

The illustration of the batch selection is depicted in Fig. 2.9. This mechanism is also called a **dispatching scheduling rule**, where the idea is to assign priorities to the incoming tasks and the tasks with higher priorities win (they get allocated to the resources faster).

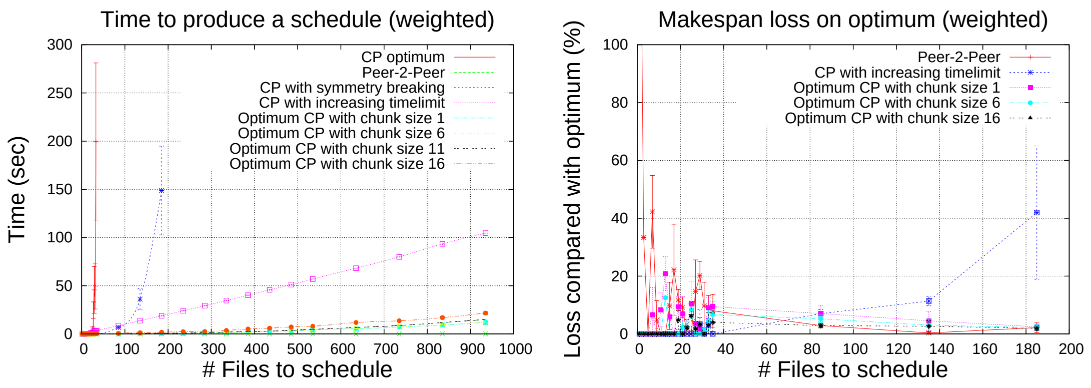


Figure 2.8: Approximation of the solver's runtime depending on different strategies (left) and corresponding loss of the makespan comparing to an optimal schedule (right) for the weighted case.

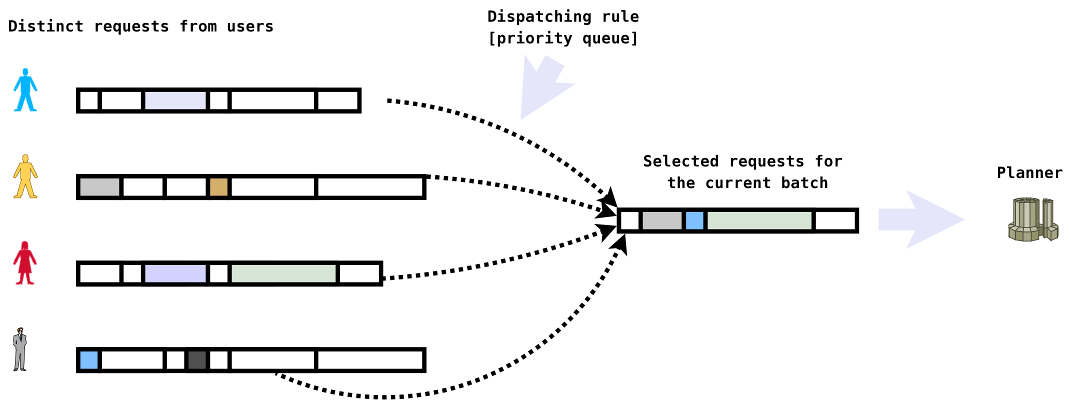


Figure 2.9: Illustration of a dispatching process. Depending on several factors, the transfers from distinct users are selected for the current batch.

Dispatching rules do not have to lead to the global optimality, but generally provides very sufficient results and fit well into a dynamic environment with the requirements for a high throughput (also used in routers, grids, etc.).

As we know, requests can be made by many users and asking for several different datasets spread over many locations. These requests are naturally dis-organized (ahead of the time) affecting an overall performance and a delay of delivery with respect to the users. The ultimate goal of a dispatching rules is to "organize" requests according to several criteria and deliver a sustained performance along the maximal quality of service where all users have ideally identical allocations of the provided service (i.e. fair-share for users). The criteria are for example parameters influencing the performance in order to accomplish the sustained data throughput, but also user's importance (e.g. priority) determining how the system's allocation should be distributed among users. Generally, such system has to present a strategy fulfilling many requirements.

If we look back at the Fig. 2.9, where there are incoming requests from multiple users, the most simple dispatching rule can be **FIFO** (**F**irst **I**n **F**irst **O**ut) [34]. In this case, the requests are ordered exactly as they appear in the system and no other attributes are taken into an account. Introducing the priorities (i.e. importance of a request) to different users can lead to assigning a different weights to their particular tasks, consequently give birth to the **Weighted Fair Queuing** (WFQ) rule. This rule (WFQ) is a generalization of a **Round Robin** (RR), where users are considered equal and their tasks are picked evenly. RR strategy is one of the simplest algorithms, resources are assigned to tasks in equal portions and in circular order, without priority (also known as cyclic executive).

Clearly, more attributes (requirements) we want to involve into the dispatching rules, more likely they will be "conflicting" (e.g. preferring higher throughput can lead to unfairness between users, and vice versa). Most often the evaluation function looks like a linear combination of the desired factors (F_i) with their appropriate weights (W_i):

$$P_f = \sum_{i=1}^{N} W_i \cdot F_f^i, \text{ where } \sum_i W_i = 1 \tag{2.1}$$

As an example, we can look into the DataCarousel, a system handling the requests for the tape system in the STAR experiment. The tape system works as a tertiary storage, where the medium used to store files are magnetic tapes handled by a robotic arm. One of the most costly aspects of dealing with robotic tertiary storage system is the time it takes to switch a tape. Another latency problem is searching for a file on a tape depending on the tape size and the search speed and also a size of the file consuming the tape. Working toward avoiding or minimizing these delays results in a large performance gain. The dispatching rule which was studied in STAR [30] led to the following parameters/factors in an evaluation function:

$$P_f = W_{switch} \cdot F_f^{switch} + W_{size} \cdot F_f^{size} + W_{usage} \cdot F_f^{usage} \tag{2.2}$$

In this case, the W_{switch} is a constant factor (weight) characterizing the importance of a tape switches, W_{size} an importance of a file size, and W_{usage} an importance of a usage history. Similarly, the corresponding attributes are:

- F_f^{switch} - number of files from the same tape
- F_f^{size} - the size of a requested file

- F_f^{usage} - usage history of a user who submitted the request

In our system, the dispatching rule is a modular component, independent of the planner itself. It operates independently with the data stored in a database. Therefore any rule can be plugged into the system, supposing all required factors are provided by the database.

2.5 Cache policy

Cache policy controls how much and which data are kept at cache space and when flushing occurs. The idea for keeping data within a cache is that future request for that data can be served faster. One can clearly see that if files otherwise available only at tape system (with long response time) are duplicated at the cache, the data delivery can be speed up.

Cache algorithms (also called replacement policies) choose which items to discard to make rooms for new ones when cache is full. The "hit-rate" of a cache describes how often a searched-for item is actually found in the cache. Since it is generally impossible to predict how far in the future information will be needed, the replacement policies are based on experience of access patterns which have locality of reference [18]. We will outline several most widely used replacement algorithms.

Least/Most Recently Used (LRU, MRU) LRU [31] is based on discarding the least recently used items. The implementations required keeping "age-bits" (may be expensive), keeping track of what was used when in order to find the least recently used items.

In contrast to LRU, MRU [16] discards the most recently used items. For random access patterns and repeated scans over large datasets MRU cache algorithms have more hits than LRU due to their tendency to retain older data.

Random Replacement (RR) and Least Frequently Used (LFU) Randomly selecting a candidate file to discard if space is needed doesn't require keeping any information about the access history. In contrast to LRU and MRU where the information considered for discarding was age of the file, LFU algorithm counts how often file is needed. The ones which are used least often are deleted first.

There also exists the *Adaptive Replacement Cache* (ARC) [38] which constantly balances between LRU and LFU to improve combined results.

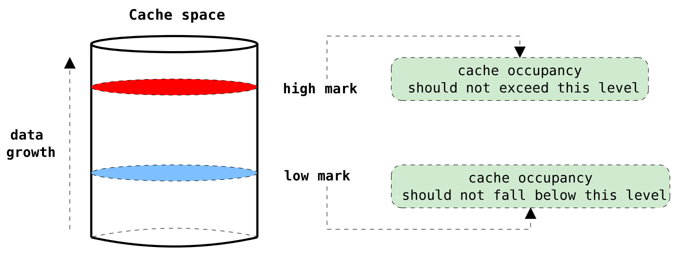

Cache space
data growth
high mark
low mark
cache occupancy should not exceed this level
cache occupancy should not fall below this level

Figure 2.10: Illustration of High-low water marking principle.

Multi Queue Caching Algorithm (MQ) It often turns out that considering only the age of the item may not be often enough for effective replacement strategy. Zhou, Philbin and Li in [74] introduce MQ algorithm which considers:

- different cost of files: keep files that are expensive to obtain, e.g. those that take a long time to get (from slow sources like MSS).
- size of files (taking up more cache): if files have different sizes, the cache may want to discard a large file to store several smaller ones.
- expiration time of files: keep information that expires and consequently delete files that are expired.

2.5.1 Water marking

The second role of the cache policy is to decide when the cache should be flushed and how many items should be deleted. In it's simplest form the cache may be completely flushed when there is no more space for a new element; however more sophisticated strategies are usually used. **Water mark** algorithms are widely implemented in cache management routines. There are two thresholds, low and high mark (see Fig.2.10), that represent the percentage cache occupancy. The low mark states that cache should always contain some amount of files and high mark states up to which level the cache may be maximally filled. For example if a low mark is set to 40% and a high mark to 70%, the algorithm tries to keep the cache level between these two percentage levels. In other words, discarding of files is activated and deactivated depending on these thresholds.

There exist also adaptive algorithms [39], where the thresholds are dynamically adjusted according to the varying I/O workload. Two thresholds are defined as the multiplication of changing rates of the cache occupancy level and the time required to fill and empty the cache.

As we will see in Chapter 4 any cache replacement strategy can be plugged into the system. Since our intent was not to study cache management, we implemented only the simple LRU rule within water marking policy.

Chapter 3

Planning problem formalization

In this chapter we will present a formal description of the problem using mathematical constraints which present restrictions from reality.

The input of the problem, which can be constructed at any point in time, consists of two parts. First one has a static character and represents the network and file origins. The network, formally a directed weighted graph, consists of a set of nodes **N** and a set of directed edges **E**. Nodes represent computing sites while edges transfer links between sites. The weight of an edge corresponds to the link bandwidth, i.e. throughput of units of file size per unit of time. The information about file's origins is a mapping of that file to a set of sites where the file is available. In reality, this input part is not static since the load of the links varies with time, hence their bandwidth fluctuates as well. Moreover some links may break and become unavailable. However, for the purpose of defining a planning and scheduling model we will consider these input parts static received at the beginning and which do not change during the solving process.

The second part of the input is a user request, namely the set of files that are going to be transferred and their destination site. The goal of the solver is to produce:

- a transfer path for each file, i.e. selection of one origin and a valid path starting from the origin node and leading to the destination,
- for each file and its selected transfer path, allocation of particular link transfers in time, such that
- the resulting plan has minimum makespan (the finish time of the last transfer).

The solving process is composed of two iterative stages and we will describe each of them separately continuing with the explanation of their interaction and possible reduction of search space exploration.

3.1 Constraint programming approach

An alternative approach to programming which relies on a combination of techniques that deal with **reasoning** and **computing** is called **constraint programming** [48, 17]. The central notion is that of a constraint - a relation over the domains of sequence of variables. One can view it as a requirement that states which combinations of values from the variable domains are admitted. In turn, a **constraint satisfaction problem** (CSP) consists of a finite set of constraints, each on a subsequence of variables.

To solve a given problem by means of constraint programming we first formulate it as a constraint satisfaction problem. This part of the problem solving is called **modeling**. In general, more than one representation of a problem as a CSP exists. Then to solve the chosen representation we use mostly the general methods, which are concerned with the ways of reducing the search space and with specific *search methods*. The algorithms that deal with the search space reduction are usually called *constraint propagation algorithms*. They maintain equivalence while simplifying the considered problem and achieve various forms of *local consistency* that attempt to approximate the notion of (global) consistency.

In practice we are interested in:

- determining whether the chosen representation has a solution (is consistent)
- finding a solution, respectively, all solutions
- finding an optimal solution, respectively, all optimal solutions w.r.t. some quality measure (objective function)

The basic characteristics of constraint programming are:

- **Two phases approach:** The programming process consists of two phases: a generation of a problem representation by means of constraints and a solution of it.
- **Flexibility:** The representation of a problem by means of constraints is very flexible because the constraints can be added, removed or modified. This flexibility is inherited by constraint programming.

Problems that can be best solved by means of constraint programming are usually those that can be naturally formulated in terms of requirements, general properties, and for which domain specific methods lead to overly complex formalizations.

3.1.1 Planning stage

The aim of planning stage is to generate a valid transfer path for each requested file from one of its origins to the destination node. The following formalism is used for defining a model in a mathematical fashion.

The set $\mathbf{OUT}(n)$ consists of all edges leaving node n, the set $\mathbf{IN}(n)$ of all edges leading to node n. Input received from a user is a set of file names needed at the destination site **dest**. We will refer to this set of file names as to demands, represented by **D**. For every demand $d \in D$ we have a set of sources $\mathbf{orig}(d)$ - sites where the file d is already available.

We will present two approaches, namely *link-based* and *path-based*, for modeling planning constraints.

Shared links - constraints or nodes? At the beginning we thought about using cumulative resources in CP model for modeling a shared link or router at some site. To fully model the layout of the real network is impossible and the model is just an approximation. We decided to model the shared parts if necessary using *dummy* nodes and respected bandwidth on *dummy* links. This allows us to keep the model simple and if needed only extend the input graph.

Modeling with binary variables An important and very common use of $0-1$ variables is to represent binary choice. Let's consider an event that may or may not occur, and suppose that it is part of the problem to decide between these two possibilities. To model such a dichotomy, we use a binary variable x and let

$$x = \begin{cases} 0 & \text{if the event occurs} \\ 1 & \text{if the event doesn't occur} \end{cases}$$

The event itself may be almost anything, depending on the specific situation being considered.

Link-based approach.

The essential idea for this principle is to use one decision variable for each demand and link of the network (edge in a graph). We will refer to this $\{0,1\}$ variable as X_{de}, denoting whether demand d is routed (value 1) over the edge e of the network or not (value 0).

Mathematical constraints, ensuring that if all decision variables have assigned values the resulting configuration contains transfer paths, are introduced below.

$$\begin{gathered}\forall d \in \mathbf{D}: \\ \sum_{e \in \cup \mathbf{OUT}(n|n \in \mathbf{orig}(d))} X_{de} = 1, \quad \sum_{e \in \cup \mathbf{IN}(n|n \in \mathbf{orig}(d))} X_{de} = 0\end{gathered} \tag{3.1}$$

$$\forall d \in \mathbf{D}: \sum_{e \in \mathbf{OUT}(dest(d))} X_{de} = 0, \quad \sum_{e \in \mathbf{IN}(dest(d))} X_{de} = 1 \tag{3.2}$$

$$\begin{gathered}\forall d \in \mathbf{D}, \forall n \notin orig(d) \cup \{dest(d)\}: \\ \begin{array}{rl} \sum_{e \in \mathbf{OUT}(n)} X_{de} & \leq 1 \\ \sum_{e \in \mathbf{IN}(n)} X_{de} & \leq 1 \end{array} \quad \sum_{e \in \mathbf{OUT}(n)} X_{de} = \sum_{e \in \mathbf{IN}(n)} X_{de}\end{gathered} \tag{3.3}$$

The *path constraints* (Eq. (3.1), (3.2), (3.3)) state that there is a single path for each demand. The demand must leave exactly one of its origins and cannot be transferred to a site where it already is (Eq. (3.1)). It has to enter the destination site from exactly one of its incoming links and once it is there it cannot leave it (Eq. (3.2)). Each demand must enter some site by at most one link (the same holds for leaving) and if it enters some site it must also leave it (Eq. (3.3)). These constraints alone allow isolated loops along with the valid paths and therefore *precedence constraints* are used to eliminate such loops.

Precedence constraints (Eq. (3.4)) use non-decision positive integer variables P_{de} representing possible start times of transfer for demand d over edge e. Let dur_{de} be the constant duration of transfer of d over edge e. Then the precedence constraint

$$\begin{gathered}\forall d \in \mathbf{D}\ \forall n \in \mathbf{N}: \\ \sum_{e \in \mathbf{IN}(n)} X_{de} \cdot (P_{de} + dur_{de}) \leq \sum_{e \in \mathbf{OUT}(n)} X_{de} \cdot P_{de}\end{gathered} \tag{3.4}$$

ensures a correct order between transfers for every demand, thus restricting loops. Unfortunately, constraints (Eq. (3.4)) do not restrict the domains of P_{de} until the values X_{de} are known and therefore we suggest using a redundant constraint to estimate better the lower bound for each P_{de}. Let $start$ be the start vertex of e not containing demand d ($start \notin orig(d)$):

$$\min_{f \in \mathbf{IN}(start)} (P_{df} + dur_{df}) \leq P_{de} \tag{3.5}$$

Variables P_{de} can be used not only to break cycles but also to estimate makespan of the plan as shown in Section 3.1.5

Path-based approach.

Similarly to the *link-based* model, we use X_{de} variable for every demand and link, but in this case, the X variables are not decisional. Instead, we generate all possible paths **P** from each node to the destination $dest$. For every demand d and path $p \in P$ a $\{0,1\}$ decision variable P_{dp} is introduced. Its assignment to 1 means the file will be transferred over path p, 0 means the opposite. If we denote the set of all possible paths from site n to destination $dest$ as $\mathbf{OP}(n)$, then the following constraint ensures that each file leaves exactly one of its origins using a single path:

$$\begin{gathered}\forall d \in \mathbf{D}: \\ \sum_{p \in \cup_{n \in \mathbf{orig}(d)} \mathbf{OP}(n)} P_{dp} = 1, \quad \sum_{p \notin \cup_{n \in \mathbf{orig}(d)} \mathbf{OP}(n)} P_{dp} = 0\end{gathered} \tag{3.6}$$

The assignment of X variables is provided automatically via constraint propagation by a constraint defined as:

$$\forall d \in \mathbf{D}, \forall e \in \mathbf{E}: \ X_{de} = \sum_{p \mid e \in p} P_{dp}, \tag{3.7}$$

stating that demand d is transferred via edge e if and only if it is transferred by one of the paths incident to e.

3.1.2 Scheduling stage

The goal of the scheduling stage is to evaluate the path configuration in the sense of the required makespan. Essentially, it works as an objective function because the realization of the schedule will not depend on particular transfer times calculated in this phase, as we will show in Section 4.1.

We will use the notion of **tasks** and **resources** from the area of scheduling. For each link e of the graph that will be used by at least one file demand, we introduce a unique **unary resource** $\mathbf{R}_e$. Similarly, for each demand and its selected links (defining a transfer path) we introduce a set of tasks in the following way (depicted in Figure 3.1):

- if demand d should be transferred via link e (i.e. $X_{de} = 1$) we will create task $\mathbf{T}_{de}$, encapsulating a positive integer variable $start_{de}$ (with domain $[0, \ldots, horizon]$) and constant dur_{de}, representing starting time and a duration of the transfer respectively. We will assign T_{de} to resource $\mathbf{R}_e$

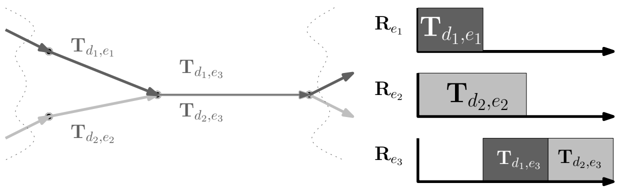


Figure 3.1: Example of assigning tasks (file transfers over particular links) into unary resources (links). In this case the transfer paths of demands d_1 and d_2 share link e_3, and consequently resource $\mathbf{R}_{e_3}$ as well.

- for any two demands d_1 and d_2 assigned to resource R_e, the constraint $start_{d_1e} + dur_{d_1e} \leq start_{d_2e} \vee start_{d_2e} + dur_{d_2e} \leq start_{d_1e}$ must hold

- for each demand d we construct precedences among tasks T_{de} in such a way that a file transfer from a site ($T_{d,out}$) can start only after the previous file transfer leading to this site ($T_{d,inc}$) has finished, i.e. $start_{d,inc} + dur_{d,inc} \leq start_{d,out}$

The idea behind using unary resources instead of *energetic* ones, which seem more appropriate, is their availability in current constraint solving frameworks. However, this statement needs a proper elaboration and study of assurance that using this approach will not cause an inefficiency. We reserved the explanation and measurements to the next sections.

An objective is to minimize the makespan of a schedule, i.e. the latest finish time of the tasks.

3.1.3 Complexity of the problem

We will show that computational complexity of the problem, in particular of the scheduling stage which is strongly NP-hard. Primarily, let's explicitly state an instance for the scheduling phase. The input consists of:

- the directed weighted graph (eventual loops are allowed), where the weight of an edge determines its *throughput* and

- the set of paths without loops, all of them leading to the same vertex

Each path needs time defined by a *throughput* of a particular edge for crossing it (for a given edge, this time is identical for all paths). Crossing an edge cannot be interrupted and only one path can cross an edge in a given time. The path must cross its edges in

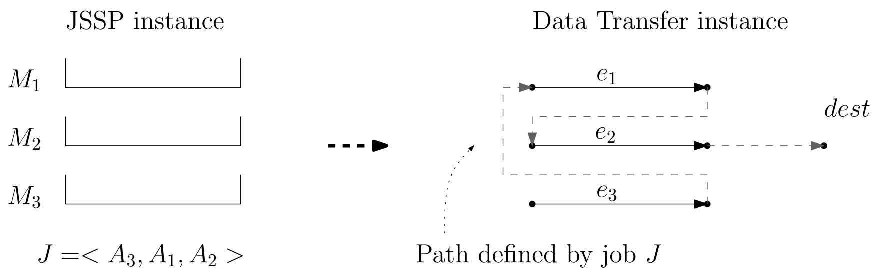


Figure 3.2: The job $J =< A_3,A_1,A_2 >$ defines a transfer path to the *dest* vertex using links e_3, e_1, e_2, with order given by precedences of actions. Connections between links (and *dest* vertex) are achieved by *dummy* edges (dashed lines) which have *slowdown* equal to 0, thus not affecting allocation times on *real* edges.

an order defined by the path itself. The goal is to allocate *edge crosses* for every path in such a way that time when the last path reaches destination vertex is minimized.

We will use the well-known three field classification $\alpha|\beta|\gamma$ for scheduling problems as described in [26]. We will demonstrate a possible polynomial-time reduction from $J3|p_{ij} = 1|C_{\max}$, a 3-machine unit-time Job-shops scheduling problem (JSSP). The transformation of the JSSP instance is following:

- for every *machine* M_1,M_2, and M_3 a unique link e_1, e_2, and e_3 is created
- a single destination site *dest* is created
- every job with ordered *actions* defines a transfer path with alternating *dummy* edges, as depicted in Fig. 3.2.
- *slowdown* for each *dummy* edge is set to 0, e.g. edge is not causing delays to any transfer. The role of *dummy* edges is to provide correct paths for each job.

The transformation written above can be realized in a polynomial time, as the size of a transformed instance increases linearly by the factor of 3 caused by *dummy* edges. Due to the fact that the $J3|p_{ij} = 1|C_{\max}$ problem belongs to the strongly NP-hard class [61], the direct consequence implies that computational complexity of our problem is at least the same.

3.1.4 Unary resources

The real network links behave more like elastic resources, where the time needed for a file to be transferred depends on the current situation and the load of a resource. Due to the lack of elastic resources in available CP frameworks we decided to model it using unary resources which brings also several simplifications (e.g. faster and more efficient filtering algorithms). This approximation seems to be accurate enough to estimate the total transfer time in a complex network environment. We verified this assumption by executing several measurements on the network links, as explained in the following text. The real transfers are realized in a “parallel” fashion to achieve a proper bandwidth saturation.

Multi-stream vs. parallel instances transfer study

The purpose of the following measurements is to demonstrate that saturation of the network link can be achieved either by running several instances of a transfer tool in parallel, or by running a single instance in a multi-stream mode. While in a first case, each instance is transferring an independent portion of data (separate files), in a multi-stream mode an instance is transferring a single data set in time, thus behaving as a unary resource.

The reason why one uses parallel methods for sending data lies in the operation of the underlying *TCP* (Transmission Control Protocol). It is a reliable stream delivery service that guarantees delivery of a data sent from one host to another without duplication or losing data. Since packet transfer over the network is not reliable, a technique known as positive acknowledgment with retransmission [1] is used to guarantee reliability of packet transfers. This fundamental technique requires the receiver to respond with an acknowledgment message as it receives the data. The sender keeps a record of each packet it sends, and waits for acknowledgment before sending the next packet. The sender also keeps a timer from when the packet was sent, and retransmits a packet if the timer expires. The timer is needed in case a packet gets lost or corrupted. One can easily see that the time for transferring the real data is just a part of the overall time the sender needs for communication with a receiver. This overall time depends (expands) on the characteristic of the network, especially **r**ound **t**rip **t**ime (*RTT*), distance and the packet loss.

The measurements were done using three different links, varying in a distance of end points, routing and *RTT*. The transfer tool used for all the measurements was *iperf* [2]. We used the default *TCP* window size set to 64 Kb. One can calculate the window size for the best performance according to the formula $\mathrm{RTT} * \mathrm{desired\_bandwidth}$ ([42]). However, since we are interested in comparing multi-stream versus parallel-instance transfer, the environment will be consistent for the comparison and there is no need for adjusting it now.

[1]TCP protocol specification - RFC 793

[2]Iperf: http://sourceforge.net/projects/iperf/

Prague local link The link is between *Bulovka* and *Golias*, two local laboratories in Prague. The link is dedicated with a static routing, with a maximum bandwidth $1Gbit/s$ and ≈ 0 RTT (optical fiber used over LAN). The endpoints used:

- `dfpch.ujf.cas.cz` (source)
- `ui5.farm.particle.cz` (destination)

BNL $\Rightarrow$ LBNL link The link is between *BNL* and *LBNL* laboratories, STAR's Tier-0 and Tier-1 site, respectively. The link is not dedicated and is routed over Esnet provider. The *RTT* for the tested endpoints is $83.354ms$. The endpoints used:

- `stargrid03.rhic.bnl.gov` (source)
- `pdsfsrm.nersc.gov` (destination)

BNL $\Rightarrow$ Prague link The link is between *BNL* and *Prague, Bulovka* laboratory, STAR's Tier-0 and Tier-2 site, respectively. The link is dedicated with a static routing, with a limited bandwidth 150 MBit/s. The *RTT* for the tested endpoints is $93.428ms$. The endpoints used:

- `stargrid03.rhic.bnl.gov` (source)
- `dfpch.ujf.cas.cz` (destination)

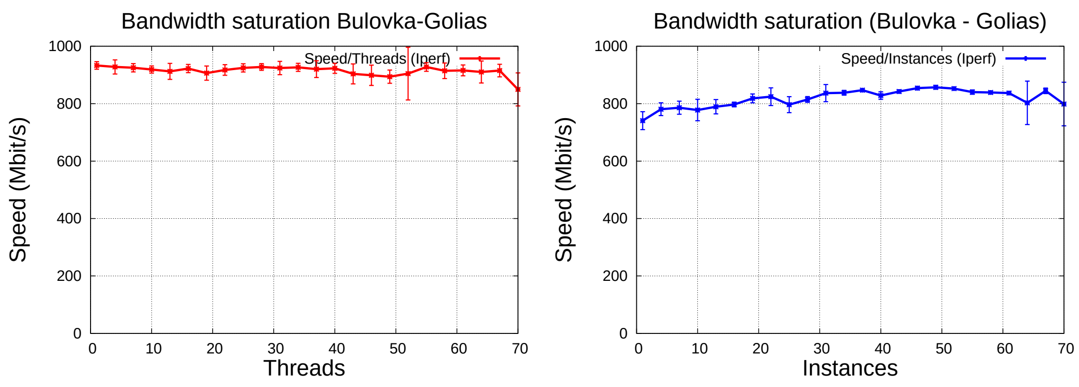


Figure 3.3: Dedicated *LAN*. **Left.** Link saturation using multiple streams in a single instance. **Right.** Link saturation using multiple instances.

In general, the outcome of the measurements is as expected. As we increase the number of streams or instances, the transfer rate and performance increase as the network bandwidth saturates. For the *LAN* transfer (Fig. 3.3), increasing the number of

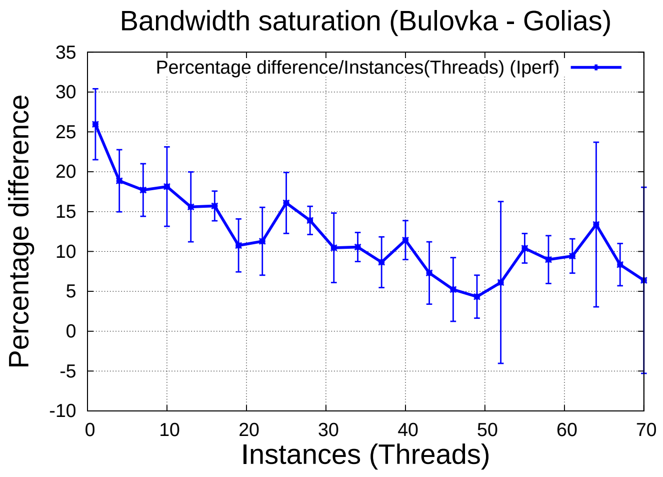


Figure 3.4: Dedicated *LAN*. Comparison of both transfer methods, Y axis showing the percentage gain in multi-stream mode over multiple instances. The positive value means the multi-stream mode outperformed multi-instance one.

streams/instances does not lead to significant speed increase since at one thread, a dedicated transfer already takes the full link speed. This is due to the negligible *RTT* and thus minimal overhead in packet acknowledgements. But moreover, it is clear from Fig. 3.3 that the thread handling does not decrease performance over single thread usage (overall transfer performance remains constant as the number of thread increases), an observation which gives confidence that we do not suffer from any other second order effects due to thread handling.

Figures 3.4, 3.6 and 3.8 represent the confrontation of both transfer methods for the three studied links. The horizontal axis displays the number of threads or running instances and the plot captures whether multiple threads (streams) method outperformed the multiple instance one. For instance value 15 means the multiple streams were in 15% better than multiple instance method, while the negative value would mean the reverse benefit. The error bars represent the standard deviation of the difference. When multiple instance transfer mode was running, we observed a lower performance comparing to handling it with threads (Fig. 3.4) and infer this decrease of performance is due to higher resource consumption and potentially, across process synchronization problems (two instances of *Iperf* do not know about each other, each making aggressive requests for resources - any operation needs to be coordinated by the OS). A several observation from the measurements are:

- Dedicated WAN (Fig. 3.7, 3.8). A reasonable case and exhibits a pattern with benefit of multi-threads up to 40.

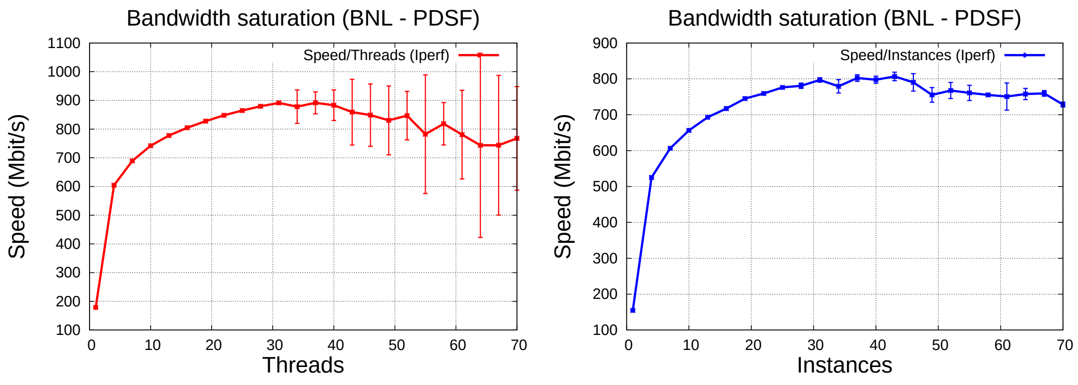


Figure 3.5: Non-dedicated *WAN*. **Left.** Link saturation using multiple streams in a single instance. **Right.** Link saturation using multiple instances.

- Not dedicated WAN (Fig. 3.5, 3.6). We can see clear benefit of multi-threads over multi-instances at lower thread count (up to 30-40).
- Dedicated LAN (Fig. 3.3, 3.4). The best case for study - no RTT, error bars and fluctuation are small over short distances allowing to best estimate the effect of the two modes without interferences or convolution from other effects.

For debating unary resources, the LAN Prague measurement (Fig. 3.4) serves as the key plot. On WAN, things are more complicated but the trend is similar and we can conclude the observations into the following:

(a) threads (streams) and sending file by file are more efficient in bandwidth saturation and cause less resource overhead than trying to send multiple files in parallel

(b) having multiple senders may bring the advantage of redundancy but this can be done from multiple sender nodes rather than multiple instances on one node (for spreading the load)

(c) link can be saturated using threads or instances (no problem either ways as far as resources are available)

3.1.5 Constraint model and solving strategy

In the previous section we have explained the main notions and the basic decomposition idea of the solving mechanism with two iterative stages. The selected solving approach is based on Constraint Programming technique, used in artificial intelligence and operations research, where we search for assignment of given variables from their domains (ranges),

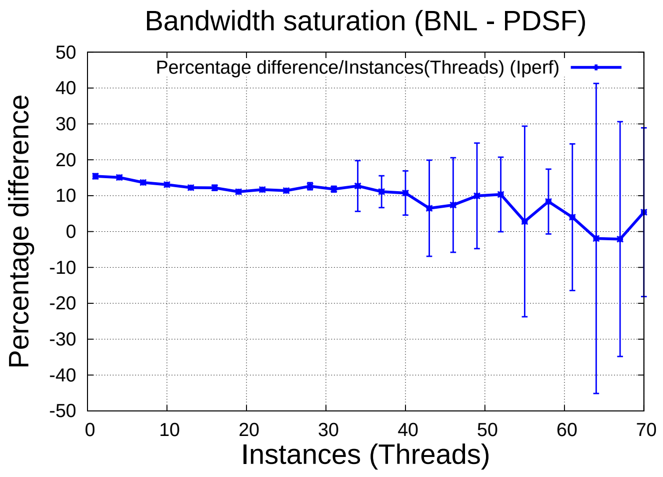


Figure 3.6: Non-dedicated *WAN*. Comparison of both transfer methods, Y axis showing the percentage gain in multi-stream mode over multiple instances. The positive value means the multi-stream mode outperformed multi-instance one.

in such a way that all constraints are simultaneously satisfied and value of an objective function is optimal.

Regarding the constraint model of the planning stage, described in the Section 3.1.1, a *link-based* model is realized directly by arithmetic constraints (Eq. (3.1) - (3.3)) and a *path-based* one by constraints in Eq. (3.6) - (3.7). Implementation of the scheduling stage, namely disjunctive constraints, is modeled by unary resource constraints with Edge Finding, Not-First/Not-Last, and Detectable Precedences filtering rules based on Theta-Lambda-tree structures [63]. The precedence constraints are implemented by inequality constraints among $start_{de}$ variables.

The principle of the search procedure and iteration of described two stage model is outlined in Alg. 1.

Algorithm 1 Pseudocode for a search procedure.

```
makespan ← sup
plan ← Planner.getFirstPlan()
while plan != null do
    schedule ← Scheduler.getSchedule(plan, makespan) {B-a-B on makespan}
    if schedule.getMakespan() < makespan then
        makespan ← schedule.getMakespan() {better schedule found}
    end if
    plan ← Planner.getNextPlan(makespan) {next feasible plan with cut constraint}
end while
```

The actual best makespan is used as a bound for scheduling phase (Branch-and-bound strategy). Moreover, the actual makespan can be effectively used also for pruning the

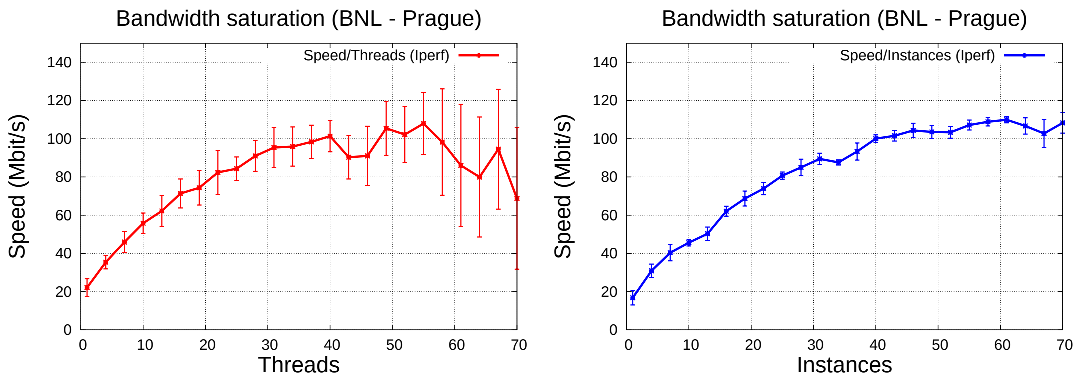


Figure 3.7: Dedicated *WAN*. **Left.** Link saturation using multiple streams in a single instance. **Right.** Link saturation using multiple instances.

search space during the first (planning) stage. The idea is that according to the number of currently assigned demands per some link and their possible starting times, we can determine the lower bound of the makespan for schedule that will be computed later in the scheduling stage. Hence if we have some upper bound for the makespan (typically obtained as the best solution from the previous iteration of planning and scheduling) we can restrict plans in next iterations by the following constraint:

$$\forall e \in \mathbf{E} : \min_{d \in \mathbf{D}}(P_{de}) + \sum_{d \in \mathbf{D}} X_{de} \cdot dur_{de} + SP_e < makespan, \tag{3.8}$$

where SP_e stands for the value of the shortest path from the ending site of e to $dest$.

Therefore as soon as the lower bound of a makespan from a currently being generated paths configuration exceeds the stored best one, the solver doesn't need to explore more branches under the current node of a search tree.

Symmetry breaking

One of the common techniques for reducing the search space is detecting and breaking variable symmetries [7]. This is frequently done by adding variable symmetry breaking constraints that can be expressed easily and propagated efficiently using ordering. One idea that can be applied in the planning stage is the following (see Fig.3.9):

- let two file demands d_1 and d_2 have overlapping sets of origins $\mathbf{O} = orig(d_1) \cap orig(d_2)$, such that $|\mathbf{O}| \geq 2$
- the set of all outgoing edges from the common origins will be denoted by $\mathbf{L}$ ($L = \cup_{n \in O} \mathbf{OUT}(n)$) and will be paired with the total order $\leq$

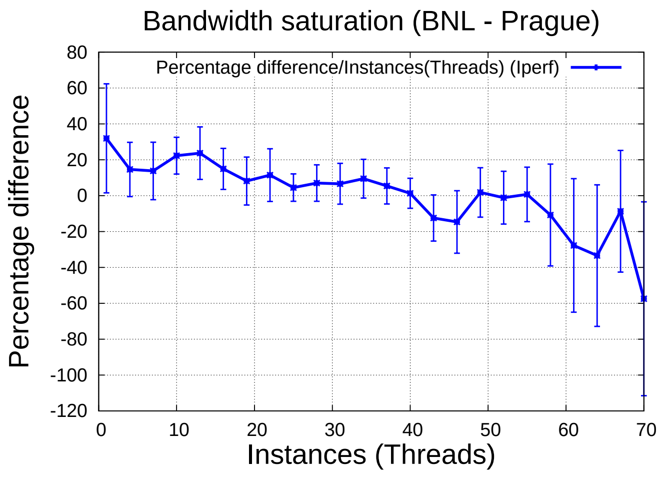


Figure 3.8: Dedicated *WAN*. Comparison of both transfer methods, Y axis showing the percentage gain in multi-stream mode over multiple instances. The positive value means the multi-stream mode outperformed multi-instance one.

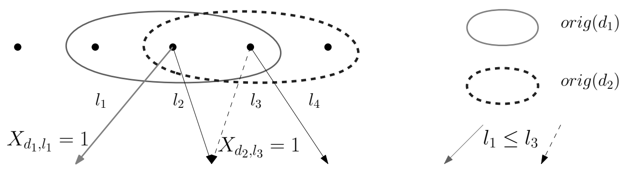


Figure 3.9: In the configuration shown, demands d_1 and d_2 have an overlapping sets of origins. The outgoing links from this intersection are l_1, l_2, l_3, and l_4. If demand d_1 is assigned to link l_1 and demand d_2 to link l_3, relation $l_1 \leq l_3$ must hold.

- if both file demands d_1 and d_2 are leaving their origins via links from the set L we will require ordering between particular links ($\forall a, b \in L : b < a \implies$ post constraint $X_{d_1,a} = 0 \vee X_{d_2,b} = 0$)

In other words if we have checked the configuration where two files are leaving their common origins by two links, it is not necessary to check also the *swapped* case .

Similar idea can be applied in the scheduling stage as well. In this case, let's suppose that several file demands are going to be scheduled for a transfer to the destination by exactly the same path. Since file demands have an identical size, we can fix the order in which files are allocated to each link from their common path. More precisely, among the corresponding tasks assigned to each unary resource representing the link from the path, we introduce additional precedences fixing their order.

Both presented methods significantly contribute to search space reduction because a real network topology and distribution of files tend to a vast amount of symmetries.

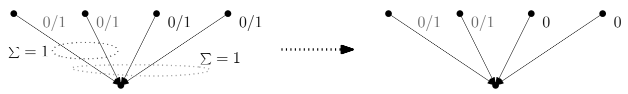


Figure 3.10: A configuration depicting a state of decision variables for some demand that is going to be transferred to the bottom destination site. If the origin of demand consists of two sites, symbolized as upper leftmost vertices, solver would still try an assignment for two rightmost links. By the filtering procedure we can reduce their values as can be seen on the right graph.

Implied constraints of X variables

The usual purpose of filtering techniques is to remove some local inconsistencies and thus delete some regions of search space that do not contain any solution. By removing inconsistent values from variable domains the efficiency of the search algorithms is improved.

A filtering principle we can apply for the decision X variables in the *link-based* planning model is following:

- let S_1 and S_2 be two sets of the decision X variables,
- let S_2 be a subset of S_1, e.g. $S_2 \subset S_1$,
- if there exists a constraint stating $\sum_{X \in S_1} X = 1$ and similarly for the second set $\sum_{X \in S_2} X = 1$, we can reduce domains of variables in $S_1 \setminus S_2$ to value 0 (depicted in Fig. 3.10).

The filtering procedure is simple. During posting of constraints into the model, each set of variables, summation of which must equal to 1, is stored. Then, search for pairs S_1 and S_2 is achieved by a quadratic nested loop over the sets stored.

Search heuristics

In constraint programming, the variable and value selection heuristics determine the shape of the search tree, which is usually traversed in a depth-first order. A clever branching strategy is a key ingredient of any constraint satisfaction approach. We have tested several combinations of variable selection and value iteration heuristics.

Initially, in the planning phase well known *dom* strategy [25] for labeling decision X variables was tested, which corresponds to selecting boolean variables in a fixed order (filtering of a variable immediately implies an instantiation). In addition, we suggested also a *FastestLink* selection for *link-based* planning approach and similarly *FastestPath* for *path-based* one. The principle for the *FastestLink* is that at the decision point from unassigned variables of demand d, the selected X_{de_j} corresponds to the fastest link ($j = \arg\min_{i=1,\dots,m} slowdown(e_i)$). If several such variables exist for demands, the first one is picked up from a fixed order. The selection principle for *FastestPath* is analogous, just the speed of a path is defined as a sum of slowdowns of its links.

According to the measurements shown in Section 3.1.6, the majority of time was spent in the planning phase, hence we proposed an improved variable selection heuristic that exploits better the actual transfer times by using information from variables P_{de}. In particular, the heuristic, called *MinPath*, suggests to instantiate first variable X_{de} such that the following value is minimal:

$$\inf P_{de} + dur_{de} + SP_e, \tag{3.9}$$

where $\inf P_{de}$ means the smallest value in the current domain of P_{de}.

Concerning the value selection heuristics, both variants were tested, particularly *Increasing* (assign first 0, then 1) and *Decreasing* (assign first 1, then 0) value iteration order.

In the scheduling phase two approaches were considered. First one, called *SetTimes*, is based on determining Pareto-optimal trade-offs between makespan and resource peak capacity. Detailed description and explanation can be found in [44]. The second one is a texture-based heuristic called *SumHeight*, using ordering tasks on unary resources. The used implementation originates from [6] and supports *Centroid* sequencing of the most critical activities.

3.1.6 Comparative studies

We have implemented and compared performance of both alternatives of the model, namely using *link-based* and *path-based* approach. Several combinations of heuristics were tried and in addition comparison with simulated Peer-2-Peer method is shown.

For implementation of the solver we use **Choco** [3], a Java based library for constraint programming. The Java based platform allows us an easier integration with already existing tools in the STAR environment.

[3]Choco: http://choco.sourceforge.net

Peer-2-Peer simulator

The Peer-2-Peer (P2P) model is well known and successfully used in areas such as file sharing, telecommunications or media streaming. P2P model doesn't allow file transfers via paths, only by direct connections. We implemented a P2P simulator by creating the following work-flow: **a)** put an observer for each link leading from an origin to the destination; **b)** if an observer detects the link is free, it picks up the file at his site (link starting node), initiates the transfer, and waits until the transfer is done. We introduced a heuristic for picking up a file as typically done for P2P. Link observer picks up a file that is available at the smallest number of sites. If there are more files available with the same cardinality of $orig(n)$, it randomly picks any of them. After each transfer, the file record is removed from the list of possibilities over all sites. This process is typically resolved using distributed hash table (DHT) [40], however in our simulator only simple structures were used. Finally an algorithm terminates when all files reach the destination, thus no observer has any more work to do.

Data sets

Regarding the data input part, the realistic-like network graph consists of 5 sites, denoted as BNL, LBNL, MIT, KISTI, and Prague and all requested files are supposed to be transferred to Prague. The distribution of files origins at one particular site, is following: the central repository is at BNL where 100% of files are available, LBNL holds 60%, MIT 1%, and KISTI 20% of all files. All presented experiment were performed on Intel Core2 Duo CPU@1.6GHz with 2GB of RAM, running a Debian GNU Linux.

Experiments

CPU time limit was used for both phases and more precisely, if the top-level search loop detects that time cumulatively spent in planning and scheduling phase exceeded 30 seconds, the search is terminated. Table 3.1 shows comparison of six combinations of search heuristics for *link-based* and *path-based* model with a Peer-2-Peer one, with an emphasis on a makespan, as quality of the result. We can see that *link-based* model gives generally better results than *path-based*, and the most efficient combination of heuristics is *FastestLink* variable selection in *Decreasing* value order with the *SumHeight* heuristic used in a scheduling phase. In this case the solver produced schedules that were better than P2P for all input instances, while the most significant benefits ($\approx$ 50% gain) are for instances up to 50 files. For instances of 40 and more files, all combinations of heuristics reached the time limit of 30 seconds. For P2P all makespans were obtained in less than 1

		Link based			**Path based**		
Files	**P2P**	*dom* $\nearrow$ *ST*	*fast* $\searrow$ *ST*	*fast* $\searrow$ *SH*	*dom* $\nearrow$ *ST*	*fast* $\searrow$ *ST*	*fast* $\searrow$ *SH*
1	1	1	1	**1**	1	1	1
20	24	12	12	**12**	77	15	15
40	40	22	22	**22**	149	33	33
60	56	42	42	**42**	237	48	48
80	72	57	57	**57**	?	64	64
100	88	73	73	**73**	?	81	81
120	104	89	89	**89**	?	104	103
140	120	103	103	**103**	?	128	127
160	144	117	117	**117**	?	150	149
180	152	137	134	**132**	?	172	172
200	168	165	165	**146**	?	193	192

Table 3.1: Comparison of makespans. $\nearrow$, $\searrow$ - Increasing, Decreasing value selection, ST - *SetTimes*, SH - *SumHeight* heuristic.

Files	number of requested files to transfer
RunTime	time in seconds taken by solver to generate result
% in 1^{st}	percentage of overall time spent in the planning phase
% in 2^{nd}	percentage of overall time spent in the scheduling phase
# of 2^{nd}	number of scheduling calls
P2P-M	time in general units to execute plan from P2P simulator
Makespan	time in general units to execute plan from CSP solver

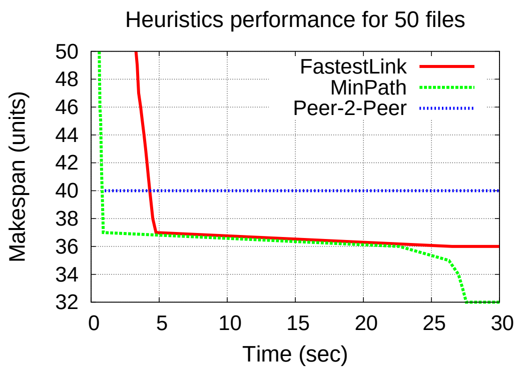
Files	**RunTime**	**% in** 1^{st}	**% in** 2^{nd}	**#of** 2^{nd}	**P2P-M**	**Makespan**
1	0.353	**70%**	30%	1	8	1
20	3.548	**77%**	23%	18	24	12
40	30	**91%**	9%	169	32	22
60	30	**98%**	2%	42	56	43
80	30	**98%**	2%	23	72	58
100	30	**98%**	2%	28	88	73
120	30	**80%**	20%	119	104	89
140	30	**95%**	5%	40	120	103
160	30	**94%**	6%	45	144	117
180	31	**92%**	8%	47	152	134
200	32	**89%**	11%	57	168	146

Table 3.2: Results for *link-based* approach with *FastestLink* selection in $\searrow$ order for planning phase and **SH** heuristic for scheduling phase.

second. Makespans are in general time units, where 1 unit in reality depends on real link speeds, and we can roughly estimate this 1 unit to the couple of seconds. Hence, the time taken to compute a schedule is paid-off by savings resulting from a better makespan.

In reality the network characteristic is dynamic and fluctuates in time. Hence, trying to create a plan for 1000 or more files that will take several hours to execute is needless, as after the time elapsed the computed plan may no longer be valid. Our intended approach is to work with batches, giving us another benefit of implementing fair-share mechanism in a multi user environment. Particularly, the requests coming from users are queued and differ in size and priorities of users. The possibility to pick demands from waiting request into batch within reasonably short intervals is very convenient for achieving fair-shareness. The experiments give us an estimate on the number of files per batch.

For a better decomposition and estimate of times spent in the phases we studied heuristics combinations such as *FastestLink* + *SumHeight* (see Table 3.2). On the ba-



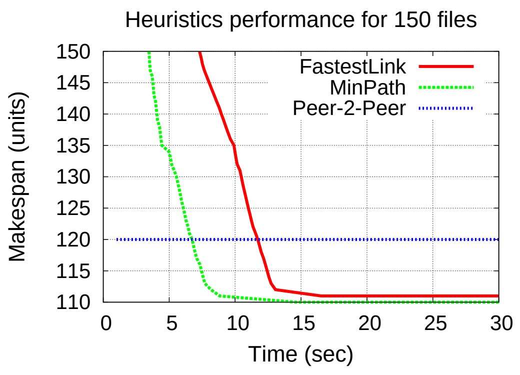


Figure 3.11: Convergence of makespan during the search process for *FastestLink* and *MinPath*.

	Solution time		**Makespan**		
Files	*FastestLink*	*MinPath*	*FastestLink*	*MinPath*	*P2P*
25	3.862	1.431	14	14	24
50	26.508	27.556	36	32	40
100	8.627	3.176	73	73	80
150	16.52	14.618	111	110	120
200	26.167	14.031	146	146	160

Table 3.3: Comparison of heuristics with emphasis on time when the best solution was found and the makespan.

sis of detailed measurements, we can see that the majority of time spent in a solving process happens in the planning stage. According to this fact and an average number of scheduling calls (5-th column in Table 3.2) the implementation of the scheduling stage seems to be fairly efficient. Therefore, we have focused on the improvements for the heuristic *MinPath* in the planning stage as proposed in Section 3.1.5 and compared the performance with the *FastestLink*. Figure 3.11 shows that convergence of the new *MinPath* heuristic is faster than the *FastestLink* and both heuristics achieve better makespan than the P2P approach.

Table 3.3 shows similar comparison of heuristics and the P2P model including the time when the best solution was found for several input instances.

3.2 Mixed Integer Programming approach

Linear programming is a method of minimizing a given linear function ($\min c^T x$) with respect to the system of linear inequalities ($Ax \leq c$) [37]. Vector x represents the variables to be determined. If all can be rational, the problem can be solved in polynomial time. However when some or all of the variables must be integer, corresponding to pure integer or Mixed Integer Programming (MIP) respectively, the problem becomes NP-complete (formally intractable).

Because of robustness of the general model, a remarkably rich variety of Mixed integer models can be used to formulate just about any discrete optimization problem [41]. They are heavily used in practice for solving problems in transportation and manufacturing: airline crew scheduling, vehicle routing, production planning, etc. The applications also include operational problems such as the distribution of goods, production scheduling, and machine sequencing.

Several algorithms from operation research are widely used for solving integer programming instances in reasonable time. Reformulation of the problem into the set of linear inequalities often involves relaxation of several constraints. In the following text we introduce the extension of the data transfer problem to the MIP problem with involved approximations.

Branch and bound This is the most widely used method for solving integer programs. The idea is to ignore the integer restriction and solve the model as though all variables were real-valued. This *LP-relaxation* provides **bound** on the best objective function value obtained, and sometimes (coincidentally) results in a feasible solution. The second aspect is **branching**. As a node is expanded, two child nodes are created in which new variable (one which didn't get integer value) bounds are added to the problem. More generally, let's consider candidate variable x_j that has a non-integer value between the next smaller integer k and the next larger integer $k+1$. The branching then creates two child nodes:

- the parent node *LP* with the new bound $x_j \leq k$
- the parent node *LP* with the new bound $x_j \geq k+1$

These new nodes (bounds) force x_j away from its current non-integer value. If the *LP-relaxation* at a node assigns integer values to all integer variables, then the solution is feasible, and is the best that can be obtained by further expansion of that node. The solution value is then compared to the incumbent and replaces it if it is better. If the *LP-relaxation* is infeasible, then the node and all of its descendents are infeasible, and it can be pruned. The search proceeds until all nodes have been solved or pruned, or until some specified threshold is meet between the best solution found and the lower bounds on all unsolved subproblems.

Branch and cut For branch and cut, the lower **bound** is again provided by the LP-relaxation of the integer program. The optimal solution to this linear program is at a corner of the polytope which represents the feasible region (the set of all variable settings which satisfy the constraints). If the optimal solution to the LP is not integral, this algorithm searches for a constraint which is violated by this solution, but is not violated by any optimal integer solutions. This constraint is called a **cutting plane**. When this constraint is added to the LP, the old optimal solution is no longer valid, and so the new optimal will be different, potentially providing a better lower bound. Cutting planes are iteratively added until either an integral solution is found or it becomes impossible or too expensive to find another cutting plane. In the latter case, a traditional branch operation is performed and the search for cutting planes continues on the subproblems.

3.2.1 Extension of the model

Since required datasets usually overlap together, we would like to minimize also the data movement of the common parts. In other words, if the same file is required by different users and the transfer paths share a link, we transfer the file on common link only once.

For this extension we have to slightly modify the constraint model, since the transfer path for a file can form a *forest* - using the terminology from the graph theory.

We denote the weight of an edge corresponding to the link bandwidth as $\mathbf{bw}(e)$ - bandwidth between two sites or average latency time for the storage elements (e.g. the time to stage the file from the tape system). The information about file's origins is a mapping of that file to a set of nodes where the file is available.

The input received from the users is a set of file names $\mathbf{F}$, where for every file $f \in \mathbf{F}$ we have a set of sources $\mathbf{orig}(f)$ - sites where the file f is already available and a set of destinations $\mathbf{dest}(f)$ - sites where the file f is supposed to be transferred.

The essential idea is to use one decision variable for each file, its destination and edge in a graph. We will refer to this $\{0,1\}$ variable as X_{fed}, denoting whether file f is routed (value 1) over the edge e of the network or not (value 0) to its destination d. Mathematical constraints (Eq. (3.10)-(3.12)), ensuring that if all decision variables have assigned values the resulting configuration contains the independent transfer paths, are analogous to the Kirchhoff's circuit laws.

$$\forall f \in \mathbf{F},\ \forall d \in \mathbf{dest}(f):$$
$$\sum_{e \in \cup \mathbf{OUT}(n | n \in \mathbf{orig}(f))} X_{fed} = 1, \quad \sum_{e \in \cup \mathbf{IN}(n | n \in \mathbf{orig}(f))} X_{fed} = 0 \tag{3.10}$$

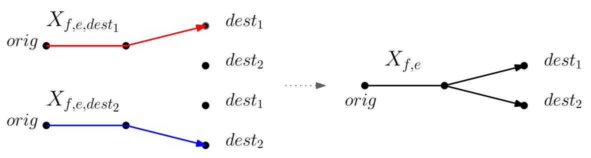


Figure 3.12: Two independent paths are *glued* together, so the file using their common links will be transferred only once (e.g. the file is staged only once, then transferred to two different destinations).

$$\forall f \in \mathbf{F},\ \forall d \in \mathbf{dest}(f): \sum_{e \in \mathbf{OUT}(d)} X_{fed} = 0,\ \sum_{e \in \mathbf{IN}(d)} X_{fed} = 1 \tag{3.11}$$

$$\forall f \in \mathbf{F},\ \forall d \in \mathbf{dest}(f),\ \forall n \notin \mathbf{orig}(f) \cup \{d\}:$$
$$\begin{array}{rl} \sum_{e \in \mathbf{OUT}(n)} X_{fed} & \leq 1 \\ \sum_{e \in \mathbf{IN}(n)} X_{fed} & \leq 1 \end{array} \quad \sum_{e \in \mathbf{OUT}(n)} X_{fed} = \sum_{e \in \mathbf{IN}(n)} X_{fed} \tag{3.12}$$

Having generated all independent paths for a file to each of its destination, we need to *glue* them together. One can look at it as creating a *forest* using the terminology from the graph theory (Figure 3.12). We achieve it by defining new binary *two-index* variable X_{fe} stating whether file f uses link e (apart from reasoning about destinations).

$$\forall f \in \mathbf{F},\ \forall e \in \mathbf{E},\ \forall d \in \mathbf{dest}(f): X_{fed} \leq X_{fe} \tag{3.13}$$

$$\forall f \in \mathbf{F},\ \forall e \in \mathbf{E}: \sum_{d \in \mathbf{dest}(f)} X_{fed} \geq X_{fe} \tag{3.14}$$

$$\forall f \in \mathbf{F},\ \forall n \notin \mathbf{orig}(f) \cup \{d\}: \sum_{e \in \mathbf{IN}(n)} X_{fe} \leq 1 \tag{3.15}$$

Finally, since we are minimizing the *makespan*, the time to transfer all files to the requested destinations, we define the constraints (Eq. (3.16)) for estimation of the completion time **T** variable and appropriate objective function: *minimize* T.

$$\forall e \in \mathbf{E}: \sum_{f \in \mathbf{F}} \frac{\mathbf{size}(f) \cdot X_{fe}}{\mathbf{bw}(e)} \leq T \tag{3.16}$$

Files sizes (identical or not)? In the former CP model we were assuming the identical file sizes, what allowed us to use particular symmetry breaking techniques to achieve more reasonable running time. The formulation of the objective function in the MIP model does not limit us in this way while the running time of the planner is still faster than the previous one. Hence, we gained the flexibility to plan in a single batch files from different datasets differing in sizes.

Planning without scheduling?

The model does not reason about the exact scheduling (timing) of file transfers in contrast to the former one explained in Chapter 3. The estimation of the total makespan is based on the idea, that this value cannot be smaller than the maximum time needed to transfer planned files over the link (in a serialized fashion - one after another, considering the full bandwidth is available for each transfer). This can be simply modeled using the formula:

$$\forall e \in \mathbf{E} : \sum_{f \in \mathbf{F}} \frac{\mathbf{size}(f) \cdot X_{fe}}{\mathbf{bw}(e)} \leq T \tag{3.17}$$

It is much faster to include this estimation already in the planning phase than reasoning about the exact schedule as we saw most of the time is spent in the later phase.

According to the simulation of the network behavior (for which we developed a standalone package) if the real transfers are realized in a greedy manner comparing to following the exact schedule we loose $\leq 3\%$ of time, which is negligible. By greedy manner we mean that as soon as the file is available at the source of any link and the plan says it has to be transferred over the link, the transfer is executed.

Benefit of this greedy approach is that it is much easier to handle inaccuracies of any kinds caused in real life scenario rather than relying on the fact that exact schedule would be always possible to fulfil. For instance, if some file would be delayed it would cause disorganizing of the next part of the schedule.

3.2.2 Implementation

The model consists of all linear constraints using *binary* (X) and *real* (T) variables. As explained in the previous section for realization of file transfers we do not need an exact schedule, only the plan (the transfer paths) that will be followed by the distributed *link managers*. Therefore, after the comparison of solving techniques we chose Mixed Integer Programming (MIP) approach which provides the most efficient results. As the backend MIP solver we use **G**NU **L**inear **P**rogramming **K**it (GLPK [23]) from Java programming language via SWIG interface ([24]).

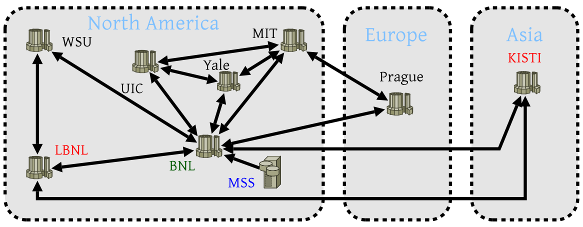


Figure 3.13: Computing centers in STAR experiment.

Files	**10**	**25**	**50**	**75**	**100**	**200**
Time (s)	0.024	0.258	0.786	1.324	2.518	9.574

Table 3.4: Average time in seconds to find optimal transfer paths.

The real-life network structure among *Tier-{0,1,2}* sites in the STAR experiment is depicted in Figure 3.13. The distribution of files is taken from empirical data, where 100% of the files are kept at MSS, 60% at LBNL, 20% at KISTI and 5% are spread among *Tier-2* sites.

According to the results (Table 3.4) planning in batches of files (to achieve adaptiveness to the network and fair-shareness to the users) seems to be realizable and payed-off by the gained optimality.

3.3 Coupling with CPUs

In the previous chapters we addressed and focused on the data transfer problem, where the task was to bring data sets to user specified locations. The role of the planner was to decide how to achieve it considering all constraints and having minimal makespan as an objective. However, very often the task is not finished by the time data are moved, but when data are analyzed. In other terms, the data movement itself only precedes the data processing.

This section discusses the extension of the model and generalizes the approach for reasoning about CPUs. We will start with reformulating the problem and introducing a few notations in an effort to cover additional computing resources and their restrictions.

Before we turn our attention into the mathematical constraints, let us underline the benefit of CPU coupling by explaining the real case with production processing in STAR (see Fig. 3.14). Part of the production is being done at Argonne computing cloud (Chicago) together with PDSF/NERSC computing center (Berkeley). The workflow is

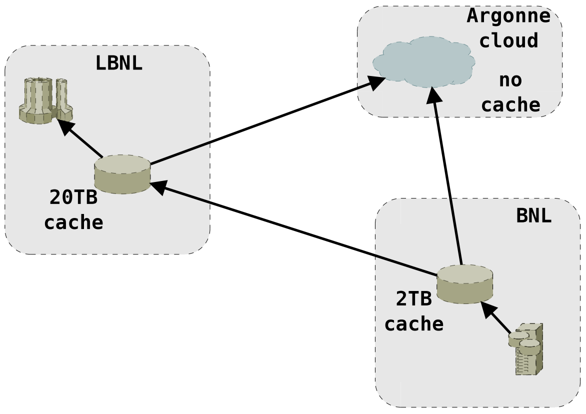


Figure 3.14: Representation of production processing using Argonne cloud with no cache space and PDSF farm with 20TB cache. Because of limited cache at BNL it turns out practical to feed Argonne cloud with data streaming from both coasts (PDSF and BNL).

following: files are continuously staged from BNL's HPSS system to the local 2TB cache. Since Argonne cloud is not equipped with any cache, the file can be transferred from BNL to Argonne only if there is a free CPU slot. To the contrary, PDSF site has sufficient 20TB cache and can hold data even if all the CPU slots are occupied. Therefore, and **this is being solved by hand**, it turns out that it is advantageous to feed Argonne's CPUs (when free) simultaneously from BNL and PDSF cache. If we had a system capable of dynamically solve this in automatic fashion the benefits could be clearly seen.

While in the pure file transfer problem the task was to locate and bring files to requested destinations, with CPU coupling the problem has to be reformulated. A user doesn't specify a single destination anymore, but a list of available processing sites along the full set of files. Each request R_i is therefore composed of set of files which need to be processed (F_{R_i}) together with a set of destinations - processing sites (D_{R_i}) where user is allowed to run jobs (Eq. (3.18)). By saying allowed we mean that user has access and can run jobs on any of these sites.

$$R_i = \{\underbrace{\{f_1, \ldots, f_{N_i}\}}_{\mathbf{F}_{R_i}}, \underbrace{\{d_1, \ldots, d_{M_i}\}}_{\mathbf{D}_{R_i}}\} \tag{3.18}$$

The task of the planner is to find transfer paths for all files (every file has to appear in one of the destinations for each request it belongs to) considering the processing phase of the file at the computing site. The system may distribute files from a request among available destinations and execute job on the portion of data set at computing site A while on the other fraction of data set at computing site B.

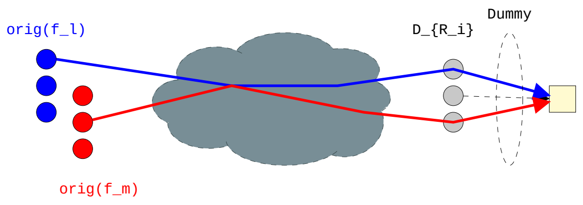


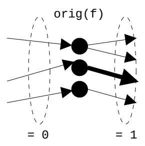
Figure 3.15: Let there be 2 files f_l and f_m belonging to F_{R_i} of some request R_i. Blue vertices represent the origins of f_l, while red ones the origins of f_m. The possible destinations D_{R_i} (processing farms) are depicted with grey vertices on the right side. First, we need to establish if there exists a transfer and processing (using *Dummy* edges) path for each file.

The graph representing the network is extended for the *Dummy* edges joining computing sites with the *dummy* vertex (Fig. 3.15). A file put on some *dummy* edge means the particular site will provide the CPU power for running user's job dependent on that file.

A computing site is represented with an average time it needs to process a unit size file. The function $\mathbf{avg}(f,e)$ takes a file f and edge $e \in Dummy$ and returns the average time it takes to process file f at site assigned to the edge e. The number of currently available (free) slots (CPUs) at the site is denoted by $\mathbf{free}(e)$. Since we will use this function later in the denominator of the relation, the return value has to be always positive. In case the site is fully occupied and there are no free slots, the function returns small $\varepsilon > 0$.

Similarly to the MIP approach for solving pure file transfer problem, the 3-index binary X variables (indexed by edge, request, and file) will form simple paths for every file respecting available origins and processing sites. Figure 3.15 visualizes this on a simple example.

We will use again mathematical constraints Eq.(3.19)-Eq.(3.22), ensuring that if all binary decision variables have assigned values the resulting configuration contains the independent transfer paths. There is a slight modification to the transfer MIP model with destination constraints, because with CPU coupling a file path has to lead to *dummy* vertex and through exactly one processing site (Eq. (3.20), (3.22)).



$$\forall i = 1,\dots,|R|,\ \forall f \in \mathbf{F}_{R_i}: \quad \sum_{e \in \cup \mathbf{OUT}(n|n \in \mathbf{orig}(f))} X_{fR_ie} = 1, \quad \sum_{e \in \cup \mathbf{IN}(n|n \in \mathbf{orig}(f))} X_{fR_ie} = 0 \tag{3.19}$$

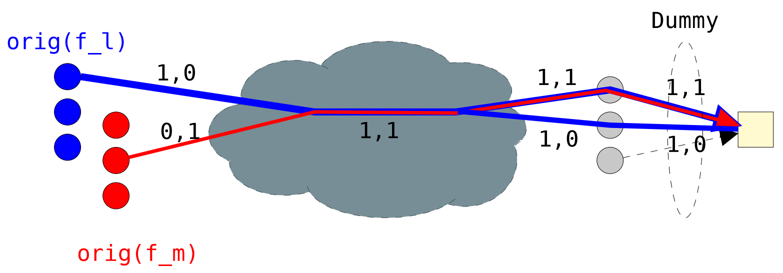
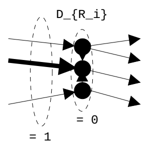
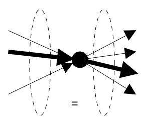


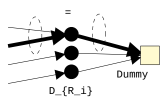
Figure 3.16: Let there are two files f_l and f_m which need to be processed by 3 independent jobs. File f_l (blue color) is sent for processing to two separate farms. File f_m (red color) is sharing the part of transfer path and processing farm with the first copy of the file f_l. Numbers over the edges indicate whether the edge participates in the file transfer (or processing) or not. First number refers to the file f_l and the second to the file f_m.



$$\forall i = 1,\ldots,|R|,\ \forall f \in \mathbf{F}_{R_i}: \quad \sum_{e \in \cup \mathbf{IN}(d|d \in \mathbf{D}_{R_i})} X_{fR_ie} = 1, \quad \sum_{\substack{e \in \cup \mathbf{OUT}(d|d \in \mathbf{D}_{R_i}) \\ e \notin Dummy}} X_{fR_ie} = 0 \tag{3.20}$$



$$\forall i = 1,\ldots,|R|,\ \forall f \in \mathbf{F}_{R_i},\ \forall n \notin \mathbf{orig}(f) \cup \mathbf{D}_{R_i}: \quad \begin{array}{rl} \sum_{e \in \mathbf{OUT}(n)} X_{fR_ie} & \leq 1 \\ \sum_{e \in \mathbf{IN}(n)} X_{fR_ie} & \leq 1 \end{array} \quad \sum_{e \in \mathbf{OUT}(n)} X_{fR_ie} = \sum_{e \in \mathbf{IN}(n)} X_{fR_ie} \tag{3.21}$$



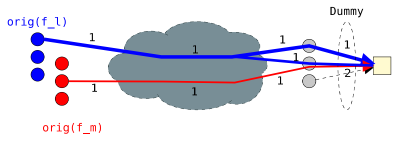
$$\forall i = 1,\ldots,|R|,\ \forall f \in \mathbf{F}_{R_i},\ \forall d \in \mathbf{D}_{R_i}: \quad \sum_{e \in \mathbf{IN}(d)} X_{fR_ie} = \sum_{\substack{e \in \mathbf{OUT}(d) \\ e \in Dummy}} X_{fR_ie} \tag{3.22}$$

The *gluing* mechanism applied to individual transfer and processing paths (marked by 3-index X variables) is similar as described in Section 3.2. However, since the processing of a file which belongs to different requests has to be considered separately (two different users' jobs are assigned to separate CPUs, even if both depend on the same file), we need to handle this in the model.

Let us first look at Fig. 3.16 depicting the sample processing case. We will first explain how variables will refer to the configuration so one can better understand what we need to achieve. There are two files f_l and f_m which need to be processed by 3 independent jobs (requests). File f_l, represented by blue color, is transferred from the origin over two links and then sent for processing to two separate farms. File f_m, represented by red color, is transferred from the origin over the first link and afterwards sharing the transfer path and processing farm with the first copy of the file f_l. Numbers over the edges indicate whether the edge participates in the file transfer (or processing) or not. In our example, the first number refers to the file f_l and the second to the file f_m. The first farm will be then processing once file f_l and once file f_m, while the second one will be processing once only the copy of f_l.

To form the *glued* transfer paths (using edges $e \notin Dummy$) into a *forest* (Eq. (3.23) - (3.25)) we introduce 2-index binary X variables stating whether the edge participates in the file transfer path (from one of its origin to the processing site). The file processing at the site is modelled using *Dummy* edges and since a single file may be processed by different jobs the 2-index X variables have to be non-negative (and not strictly binary) when dealing with edges $e \in Dummy$.

$$\forall i = 1,\ldots,|R|,\ \forall f \in \mathbf{F}_{R_i},\ \forall e \in \mathbf{E} : X_{fR_ie} \leq X_{fe} \tag{3.23}$$



$$\begin{gathered}\forall f \in \cup_{j=1,\ldots,|R|}\mathbf{F}_{R_j} : \\ \sum_{i=1}^{|R|} X_{fR_ie} \geq X_{fe} \forall e \notin Dummy, \\ \sum_{i=1}^{|R|} X_{fR_ie} = X_{fe} \forall e \in Dummy\end{gathered} \tag{3.24}$$

$$\forall f \in \cup_{j=1,\ldots,|R|}\mathbf{F}_{R_j},\ \forall n \notin \mathbf{orig}(f) \cup \mathrm{D}_{R_i} : \sum_{e \in \mathbf{IN}(n)} X_{fe} \leq 1 \tag{3.25}$$

$$X_{fR_ie} \in \{0,1\} \quad X_{fe} = \begin{cases} 0/1 & e \notin Dummy \\ 0,\ldots,|\cup_{f \in \mathbf{F}_{R_i}} R_i| & e \in Dummy \end{cases} \tag{3.26}$$

Finally, since we are minimizing the *makespan*, the time to transfer and process all files at available sites, we define two sets of constraints (Eq. (3.27)) for estimation of the completion time **T** variable and appropriate objective function: *minimize* T. First set of constraints Eq. (3.27) estimates the transfer time and the second one Eq. (3.28) the processing time.

$$\forall e \in \mathbf{E}\ \ e \notin Dummy : \sum_{f \in \cup \mathbf{F_{R_i}}} \frac{\mathbf{size}(f) \cdot X_{fe}}{\mathbf{bw}(e)} \leq T \tag{3.27}$$

$$\forall e \in \mathbf{E}\ \ e \in Dummy : \sum_{f \in \cup \mathbf{F_{R_i}}} \frac{\mathbf{avg}(f,e) \cdot X_{fe}}{\min\left(\mathbf{free}(e), \sum_{f \in \cup \mathbf{F_{R_i}}}\right)} \leq T \tag{3.28}$$

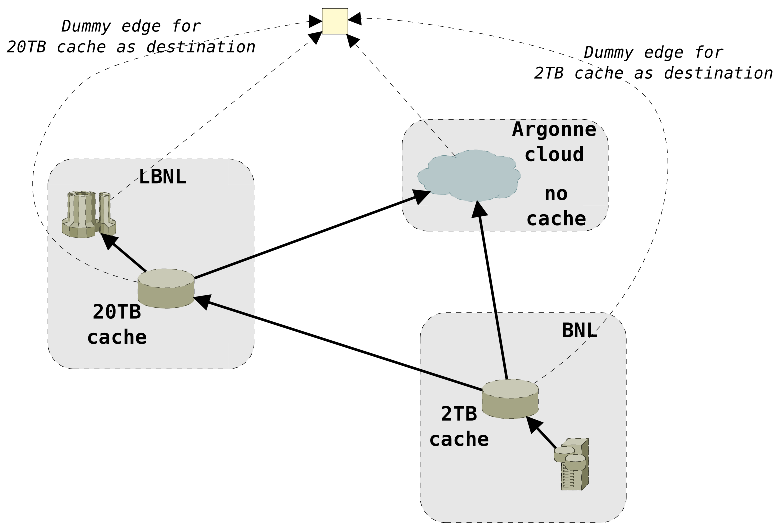


Figure 3.17: Representation of production processing using Argonne cloud as the graph input for the model. The storage cache nodes are also connected to the *dummy* destination in order to allow the model to bring files closer to the CPUs while they are taken by other jobs.

The constraint Eq. (3.27) creates a lower bound of the makespan based on necessary transfers. It simply counts the aggregated size of files assigned to transfers over each link and estimates the time it takes to move them knowing the link's bandwidth. The constraint Eq. (3.28) is a bit more complicated since the jobs processing happens concurrently on parallel CPUs. The estimate is therefore done by number of free slots and number of files to be processed on the site. Recall that function **free**(e) returns always positive value.

With the above extensions to the model we are able to address the reasoning not only about file transfers but also about file processing at different sites. However, there is one remaining issue that needs to be solved. If we look back into our initial motivation from Fig. 3.14, displaying the production processing schema in STAR, we can see that there is a substantial benefit of bringing files to the storage cache - closer to the processing sites, even if all the slots are used. The model, as it is defined above, reasons about bringing files to the processing sites since they are assigned as the only destinations. If the processing sites are occupied we would still like the solver to reason about bringing files to the storage space closer to the CPUs. In order to modify the reasoning we can create additional *dummy* links from the storages to the *dummy* destination vertex in the graph as represented in Fig. 3.17. The weights for these additional *dummy* edges need to be properly set so the solver will prioritize bringing files to the cache space in case the processing slots are taken. By setting the appropriate weight we can also control the preference between storage spaces.

Chapter 4

Technical implementation

Having described the methodology of solving approach and simulations of various alternations of the model, now we turn to technical implementation of proposed techniques. Regardless of how optimistic, universal and versatile the proposed approach may seem, even if simulations confirm the assumptions (which is the necessary step), only the functional implementation of them delivers the benefits and usefulness in the production environment. This thesis tries to provide a proper balance between the theory and practice; and this chapter presents the building blocks of the practical side.

The software design yielding the specifications, we already outlined in the previous chapters, has to address and consider many aspects. Especially in Data Grid environment, which implicitly brings distribution and loosely coupled components, it has to keep in mind **modularity**, **maintainability** and **reliability**. The software has to consist of well defined independent components which should be tested in isolation before further integration. On the other hand, grandiose and extensive design should not overwhelm its compactness as one can often see in the family of Grid tools. At the same time, the software should perform the required functions and deliver what was expected and well defined in specification.

Let us start with the software architecture with the aim to present the conceptual integrity for a system.

4.1 Architecture

It is important to pay close attention to the architecture of the system - the conceptual glue that holds every phase of a project together [14]. In this section, we will describe the elements of the system, properties and relations between them. We introduce briefly each component following the work-flow (see Fig. 4.1 for illustration).

Let us start with explaining how requests are put into the system. End users (or standalone services) generate requests using the web interface, written in *PHP* following the *MVC* design pattern. There are two possible ways how a request can be specified. **a)** either as an encapsulation of the meta-data query (as understood by STAR's File and Replica Catalogue), or **b)** providing the list of files using a *filelist*. An example of the catalogue query is:

- **production**=P10ik,
- **filetype**=daq_reco_MuDst,
- **trgsetupname**=AuAu39_production

where we specified type of the production (data set), what type of files we are interested in and some trigger setup. This meta-data query covers about $220,000$ files with a total size of 48TB. The population of the database with files belonging to the request is done asynchronously by separate component as we will see soon.

The second approach of entering the request is using a filelist, which has the following syntax:

```
SITE;STORAGE;PFN
```

Each line then describes exact location of the file given by its physical file name, the storage that holds it and finally the site where the storage is located.

The part of a request is also a desired destination for a data set (in the form of site and storage) which user selects using the web interface.

Afterwards, the request is stored in a *SQL* database (system supports *MySQL* and *PostgreSQL*) in a Catalog agnostic manner (any Catalog should work as far as they have a LFN/PFN concept our approach relies on) with the additional information like user name, group or date of the request.

Later, the component called *File Feeder* contacts the *File and Replica Catalogue* and makes the query for the requested meta-data. The output information is stored back to the database, including all possible locations for every file in a request. This is when population of the internal database with file repositories happens. Because of usually large volume of records that needs to be stored in a database, the *File Feeder* uses `LOAD DATA INFILE` syntax that provides high performance.

The main logic and reasoning about the plan happens in the brain of the system, a component called the *Planner*. It is the place where realization of the model is done and where the plan in iterations is computed. *Planner* takes a subset of all requests for files to

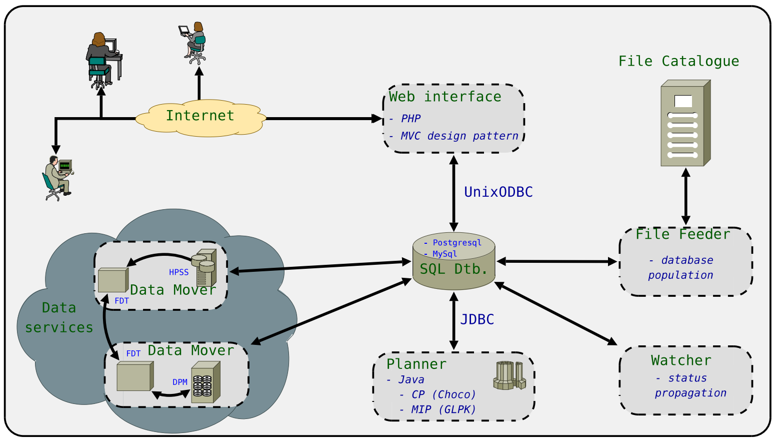


Figure 4.1: Architecture of the system.

be transferred according to the preferred fair-share function. It creates the plan (transfer paths, as we explained in the previous chapter) for the selected requests and stores the plan back to the database.

The individual file transfers are handled by the separate distributed component called *Data Mover*. As we specified at the beginning, we want to use existing point-to-point data transfer tools and use them as the back-end instruments. The *Data Mover* should serve as an intelligent wrapper on top of such tools handling the work. The role of these workers is to perform a point-to-point data transfer on a particular link following the computed plan. The results and intermediate status is continuously recorded in the database and user can check the progress at any time.

Because of the asynchronous nature of the communication between components, the system has to have well defined states and transitions from one state to another given by state diagrams. We will explain the flow in the following section. The *Watcher*, independent component running at each site, is responsible for changing states of the objects and cache management.

We can see that the whole mechanism is a combination of **deliberative** (assuring optimality) and **reactive** planning (assuring adaptability to the changing environment). Since this is crucial to the argument, we will continue with explanation of the workflow and inter component communication with the direction to the (*Planner* and *Data Mover*) serving up as a “reasoner” and a “worker”.

4.1.1 Web interface

The user interface, the space where interaction between users and system occurs, serves for providing operations, control and getting feedback from the components in a simple and compact way. It allows users to enter their requests either in the file catalogue query form or by uploading a file list, as we mentioned in the previous section. A set of screenshots from the user's web interface is shown in Fig. 4.2. System feedbacks the initial information like total size of the requested dataset, sample file paths for control and expects the confirmation from the user. If user checks and finds all information correct, the request can be confirmed. Afterwards, (when *File Feeder* populates the database, *Planner* creates first transfer paths and *Data Mover* starts executing transfers), the web interface continuously provides information about each request in the system. Among others, it includes:

- number of succeeded files (already at the destination),
- number of failed files,
- estimated time to the finish, etc.

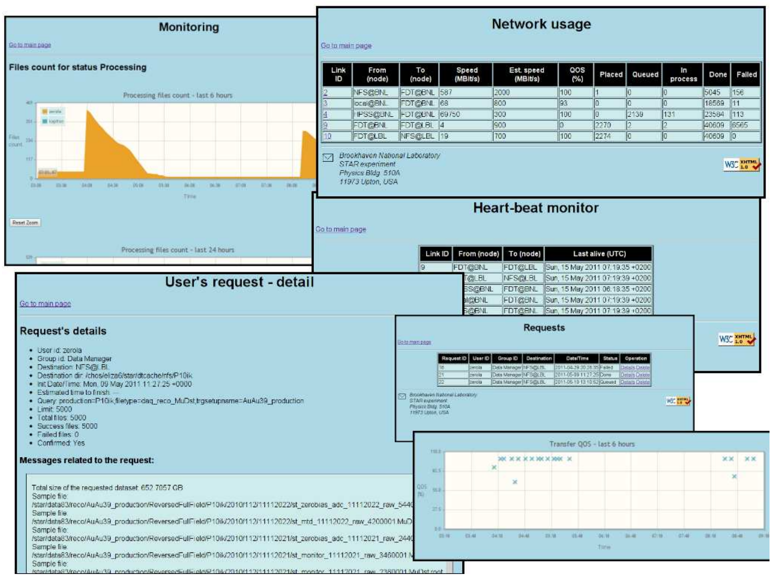
When the processing of the request is finished, the user can list the failed files and eventually resubmit only this portion again.

The interface also provides updated information for administrator/operator, where it displays the current load, speed and quality of the service per each link/service. Each resource also provides detailed graphs showing the performance and usage for past 6 hours, 24 hours and 1 week. The graphs include the history related to Quality of Service (QOS), number of files assigned to the resource in a particular state (Placed, Queued, Processed) and speed of the resource.

Keeping all distributed services up and working is often a hassle for administrators. Therefore, the web interface provides simple service monitoring, where one can check if the component is running or what was its last alive state.

The core of the web interface is written in PHP 5.2 [1] language, under the **Model-View-Controller** (MVC) architectural pattern (see Fig.4.3). The pattern isolates the application logic from the user interface (input and presentation). The purpose of the separation is that changes to the view can be implemented, or even additional views created, without having to re-factor the model. The *View* generates *XHTML* input/output and this is the only place where it can be generated. The *Controller* dispatches requests and controls flow, while the *Model* holds data representation and business logic.

[1]PHP: www.php.net


Monitoring
Go to main page
Files count for status Processing
Processing files count - last 6 hours
Reset Zoom
Processing files count - last 24 hours
Transfer QOS - last 6 hours
QOS (%)
Time

Network usage

Go to main page

Link ID	From (node)	To (node)	Speed (MBit/s)	Est. speed (MBit/s)	QOS (%)	Placed	Queued	In process	Done	Failed
2	NFS@BNL	FDT@BNL	587	2000	100	1	0	0	5045	156
3	local@BNL	FDT@BNL	68	800	93	0	0	0	18569	11
4	HPSS@BNL	FDT@BNL	69750	300	100	0	2139	131	23584	113
9	FDT@BNL	FDT@LBL	4	900	0	2270	2	2	40609	6565
10	FDT@LBL	NFS@LBL	19	700	100	2274	0	0	40609	0

Brookhaven National Laboratory
STAR experiment
Physics Bldg. 510A
11973 Upton, USA

Heart-beat monitor

Go to main page

Link ID	From (node)	To (node)	Last alive (UTC)
9	FDT@BNL	FDT@LBL	Sun, 15 May 2011 07:19:35 +0200
	T@LBL	NFS@LBL	Sun, 15 May 2011 07:19:39 +0200
	SS@BNL	FDT@BNL	Sun, 15 May 2011 06:18:35 +0200
	al@BNL	FDT@BNL	Sun, 15 May 2011 07:19:39 +0200
	S@BNL	FDT@BNL	Sun, 15 May 2011 07:19:39 +0200

User's request - detail

Go to main page

Request's details

- User id: zerola
- Group id: Data Manager
- Destination: NFS@LBL
- Destination dir: /chos/eliza6/star/dtcache/nfs/P10ik
- Init Date/Time: Mon, 09 May 2011 11:27:25 +0000
- Estimated time to finish: —
- Query: production=P10ik,filetype=daq_reco_MuDst,trgsetupname=AuAu39_production
- Limit: 5000
- Total files: 5000
- Success files: 5000
- Failed files: 0
- Confirmed: Yes

Messages related to the request:

```
Total size of the requested dataset: 652.7057 GB
Sample file:
/star/data83/reco/AuAu39_production/ReversedFullField/P10ik/2010/112/11112022/st_zerobias_adc_11112022_raw_5440
Sample file:
/star/data83/reco/AuAu39_production/ReversedFullField/P10ik/2010/112/11112022/st_mtd_11112022_raw_4200001.MuD
Sample file:
/star/data83/reco/AuAu39_production/ReversedFullField/P10ik/2010/112/11112021/st_zerobias_adc_11112021_raw_2440
Sample file:
/star/data83/reco/AuAu39_production/ReversedFullField/P10ik/2010/112/11112021/st_monitor_11112021_raw_3460001.M
Sample file:
```

Requests

Go to main page

Request ID	User ID	Group ID	Destination	Date/Time	Status	Operation
16	zerola	Data Manager	NFS@LBL	2011-04-29 20:26:35	Failed	Details Delete
21	zerola	Data Manager	NFS@LBL	2011-05-09 11:27:25	Done	Details Delete
22	zerola	Data Manager	NFS@LBL	2011-05-10 13:10:52	Queued	Details Delete

Brookhaven National Laboratory
STAR experiment
Physics Bldg. 510A
11973 Upton, USA

Figure 4.2: Several screen-shots of the Web interface.

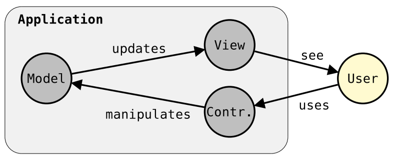


Figure 4.3: The basic MVC concept.

Couple of dynamic parts is written in *Javascript* [2], especially the module for displaying plots. For this purpose we use *jqPlot* [3], the *jQuery* plugin to generate pure client-side dynamic charts.

For accessing the SQL database we use **O**pen **D**atabase **C**onnectivity (ODBC) [4] interface which provides the translation between application and the DBMS. Because of the independence of underlying database server, the potential porting to another SQL database should be smooth.

4.1.2 Database design

This section introduces the design of an overall database system. A correct design is essential to achieving goals of the project and provides access to up-to-date, accurate

[2]Javascript: http://www.javascript.com
[3]jqPlot: http://www.jqplot.com
[4]PHP and ODBC: http://phpodbc.com

information. We will concentrate on and determine the relationship between the different data elements and the logical structure on the basis of these relationships.

When designing a database system one should try to create a proper balance between a logical design and technical optimization. On one hand, the aim of logical design is to apply Codd's rules to every table. These rules are defining what is required from DBMS in order to be considered *relational*. It is guided by rules. On the other hand, it doesn't guarantee the optimal performance of the database system, because some queries can be very complex. The technical optimization makes sure that the most important functions perform good. It is often case dependent which steps lead to higher performance of the database. Usually it is a combination of

- **denormalisation** - to avoid an expensive join in a high frequency function,
- **combining tables** - to remove a redundant table,
- **storing derived data** - to avoid repetitive time intensive computations,
- **adding indexes** - to speed up joins and look-ups for large tables,
- **partitioning** - to increase manageability, performance or availability.

The primary idea was to keep database design compact and easily manageable while having all information and functions provided. Due to the fact that STAR is exclusively using **MySQL** [5] DBMS and has experts for the performance tuning and maintenance of the servers, the decision which system to use was pre-defined. However, we aimed to build it as portable as possible and tested it also with **PostgreSQL** [6]. We use **InnoDB** transaction-safe storage engine with `FOREIGN KEY` referential-integrity constraints. Due to portability we use minimum server side triggers, only several constraints to ensure the data integrity.

The overall database schema is shown in Fig. 4.4. The tables are grouped into several categories, differentiated by colors, depending on their purpose. Some information in a database are rather *static* (although they also change in time, but due to the very low frequency of changes, we will consider them as permanent). Tables **users, groups**, and **membership** keep such information about users recognized by the system. A single user can be a member of several groups with different priorities (relationship of table users and groups via membership).

Next static table **statuses** keeps possible states in which a main user request or single transfers can be. There are five possible states:

[5]MySQL: http://www.mysql.com
[6]PostgreSQL: http://www.postgresql.org

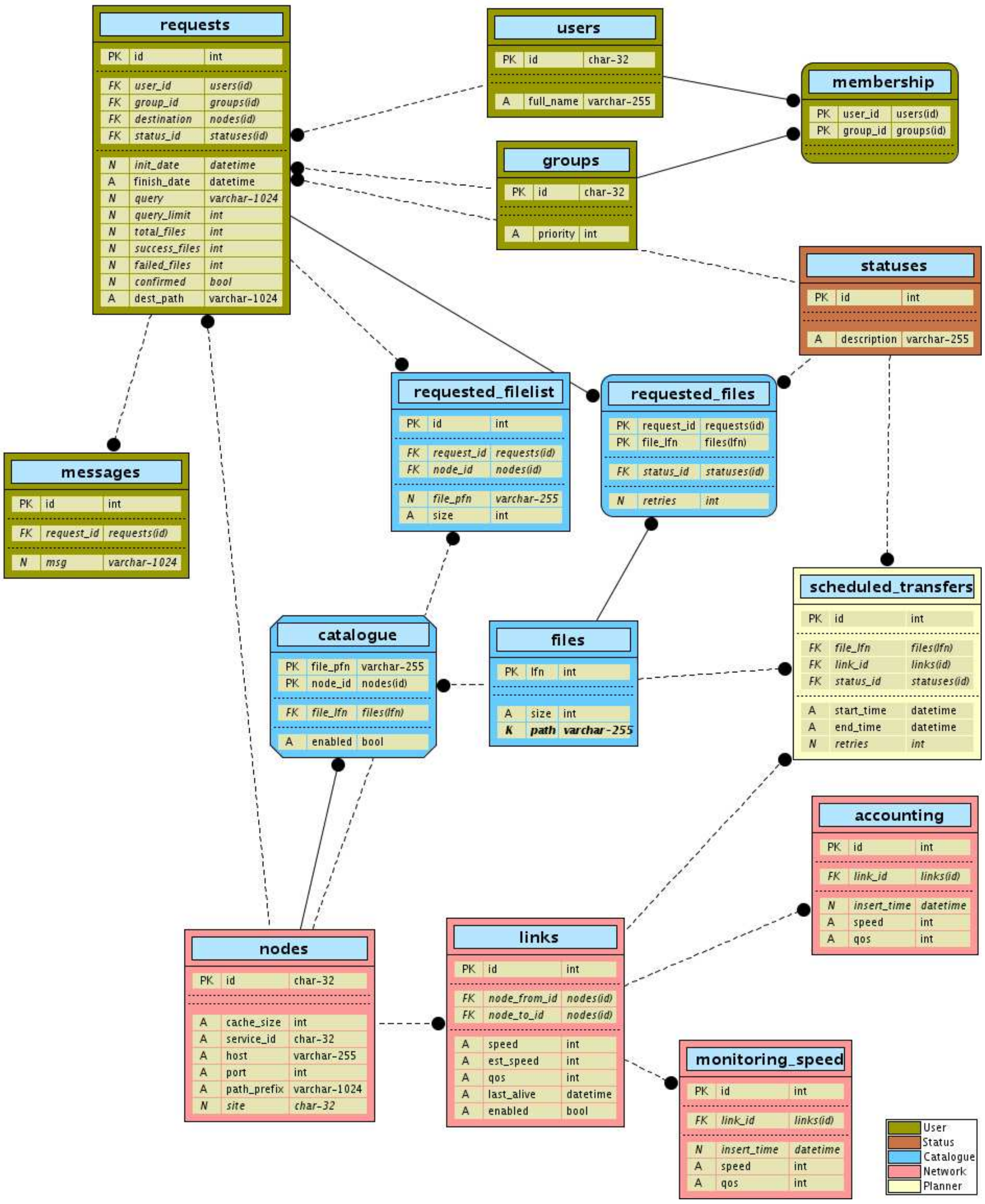


Figure 4.4: Database schema of the system outlined in entity relationship diagram.

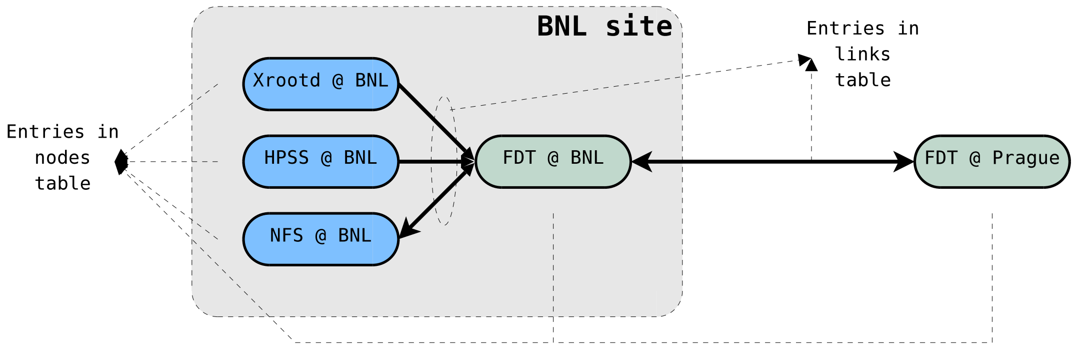


Figure 4.5: Example of a part of logical network structure.

- **Placed** - Used for user's meta data request, the initial state when the request is placed in a system.
- **Queued** - Used for user's meta data request, logical file names associated to some request, and also particular link transfers. The information states the request/file is queued in a system and waiting for processing.
- **Being processed** - Used for logical file names associated to some request, and particular link transfers. The information states the file is in a transfer.
- **Done** - Used for user's meta data request, logical file names associated to some request, and also particular link transfers. The information states the request or file transfer is successfully completed.
- **Failed** - Used for user's meta data request, logical file names associated to some request, and also particular link transfers. The information states the request or file transfer is completed, but there were errors or failures during the operation.

As we can see, statuses are used for logical requests as well as for individual file transfers. The detailed separation and the way how we store them is explained in the following sections.

The logical network structure used by the Planner and later by the Data Movers is stored in tables **Nodes** and **Links**. The first one maintains information about logical nodes, while the second one characteristics of links between them. We deliberately use term logical network, because as can be seen from Fig. 4.5, the network doesn't always correspond to the physical decomposition of the services/nodes.

We will explain the remaining parts of the database schema using flow diagrams representing the interaction of the user with the system and we will follow the flow of data throughout the system.

Incoming request

As we previously mentioned, users interact with the system using the web interface. The request including metadata catalogue query is processed asynchronously. After user submitted the request, a new record is added into the table **requests**. The *status* of the record is “Placed” and contains the requested destination for the data set, user’s id, and group’s id together with the time when the request was placed. Fields representing the number of total, success and failed files are initiated to 0. This process happens immediately while query with the catalogue and populating the database with files is an asynchronous process.

Query with the catalogue

The next step in a data flow is querying the catalogue. For this purpose, there exists a separate daemon which monitors the records in the **requests** table. If there is a placed request, the file metadata catalogue is queried with the corresponding data. Afterwards, the status of the record is changed to “Queued”. The number of total files as returned by the catalogue is updated as well. The **files** table is populated with the new logical file names and file sizes (ones not yet in a system). As stated before, several logical files from different metadata queries may overlap (Fig. 4.6). The assignment of logical file names to the metadata request is stored in the **requested_files** table, where the status of each file is initiated to state “Queued”. Finally, the table **catalogue** is populated with possible physical locations of each new file.

Planning the subset of files

As soon as the information about possible locations of files is populated in a database the *Planner* component can start its job. According to the fair-share policy (explained in Section 2.4) a subset of files (logical file names from the **requested_files** table) is selected and processed by the *Planner*. Their status is changed to “Being processed” and corresponding transfer paths are stored in **scheduled_transfers** table. A single path is stored as several records depending on the number of links the path consists of. The initial state of the individual transfer is set to either “Placed” or “Queued” depending on the position of the link. The transfer on the link outgoing from the repository holding the file (first edge in the graph) is set to “Placed” while transfers on the remaining links are set to “Queued”.

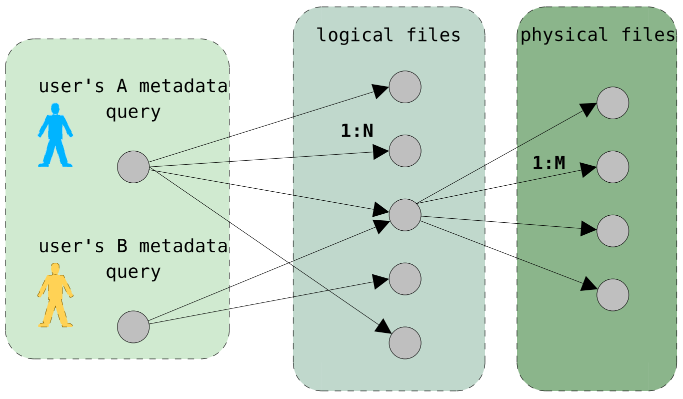


Figure 4.6: Hierarchy representing 1:N:M relationship between metadata query, logical files, and physical files. Logical files from different queries may overlap.

Transferring files

Transferring files is the core of the whole mechanism and the most complicated part from logic flow point of view. One can easily imagine that system must be able to adapt to several case scenarios, such as temporal (permanent) link failure, not responding service, etc. Each physical site is hosting a Data Mover instance, which is responsible for data transfers either in or out of the site. The Data Mover provides for each service a separate configuration. In an optimistic scenario a service executes a file transfer on its link (following the plan) and updates the status in a database so other service can work with the file, until the file is transferred to the destination. Since passing and flipping the status information is a critical part, we will first look at the flow-chart (Fig. 4.7) representing the mechanism how transfer system works with **scheduled_transfers** table.

As one can see, the system stores information from the user request in three separate levels, that creates 1:N:M relationship (Fig.4.6). At the top, there is a user's meta data request (1) which contains (N) logical files and system needs to track individual transfers for each of such file (M). Therefore, the information of success or failure of any operation from one level has to be properly propagated to the other ones.

The flow-chart from Fig. 4.7 represents information flow within the lowest level - individual transfers. The propagation from this level to the intermediate one (logical files in **requested_files** table) is handled asynchronously by the separate service called *Watcher*.

The flow-chart from Fig. 4.8 depicts the status flow within the **requested_files** table. Finally, the flow-chart in Fig. 4.9 represents information passing regarding to the top-level structure handled by the **requests** table. Similarly to the previous layer, the infor-

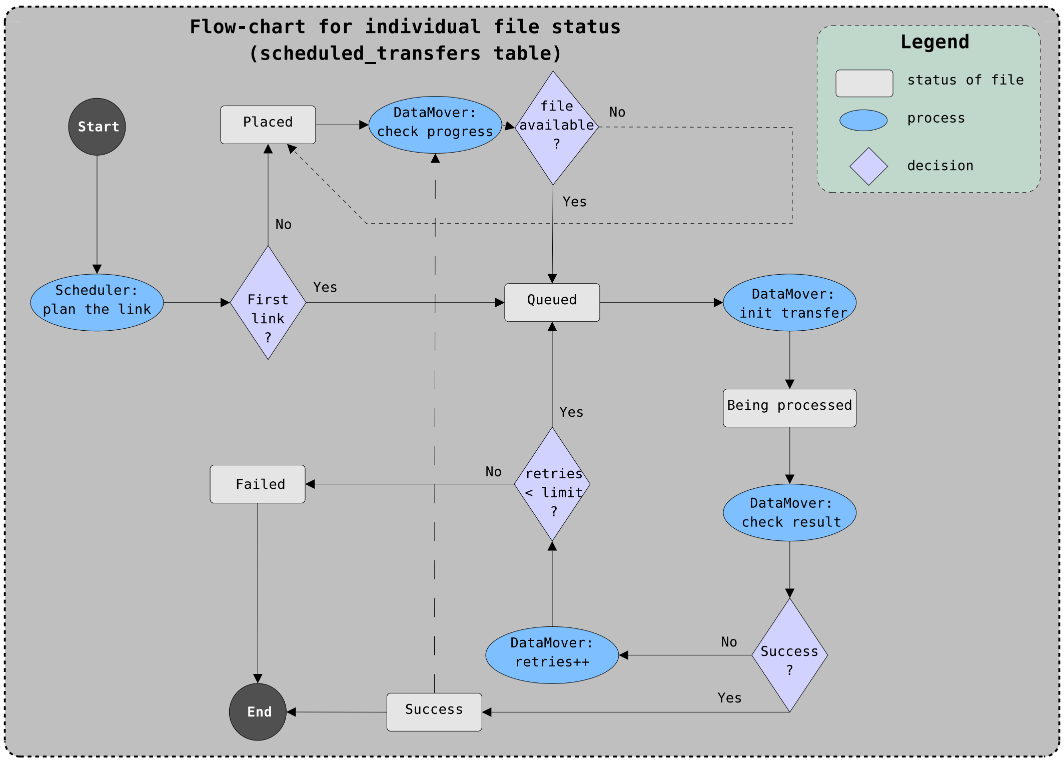


Figure 4.7: Flowchart representing the status flow related to the individual files in scheduled_transfers table.

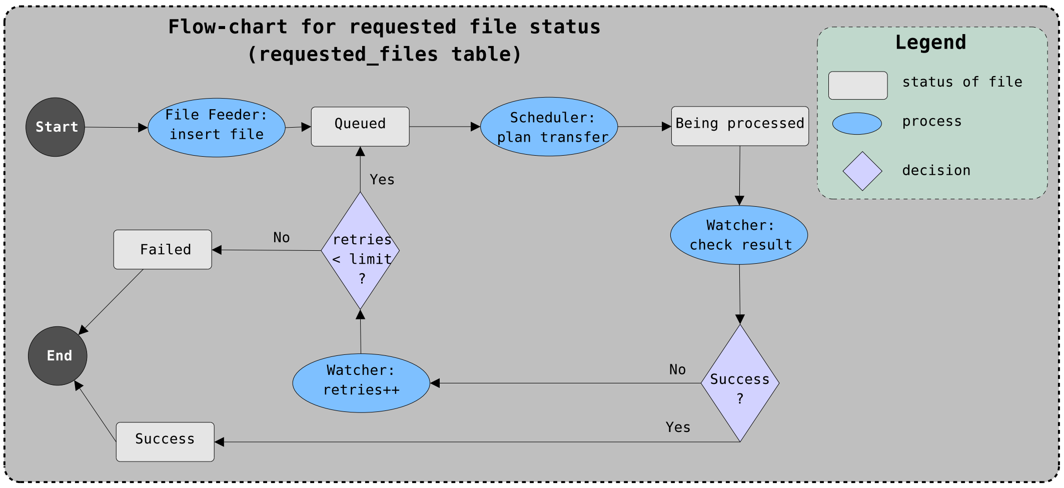


Figure 4.8: Flowchart representing the status flow related to the logical files in requested_files table.

mation propagation from the **requested_files** to meta-data queries (**requests**) is handled by the *Watcher*.

4.1.3 Watcher

As we have seen, propagating some parts of information is handled inside the Data Mover component. Nonetheless, it would be inefficient to propagate everything "in-time" due to the frequent and intensive database querying. Therefore, we have a component called *Watcher* that periodically monitors the database and performs the updates. It is written with the use of *hooks*. If there is an operation which needs to be periodically executed, it is hooked into the Watcher and information such as the frequency and additional details are put into the configuration file. There is a central Watcher running at BNL site with these following hooks:

- **accounting** and **monitoring** tables update
- **accounting** and **monitoring** tables delete
- **links** table update
- **requested_files** table update
- **cache management**

We will briefly describe what each of this hook is doing.

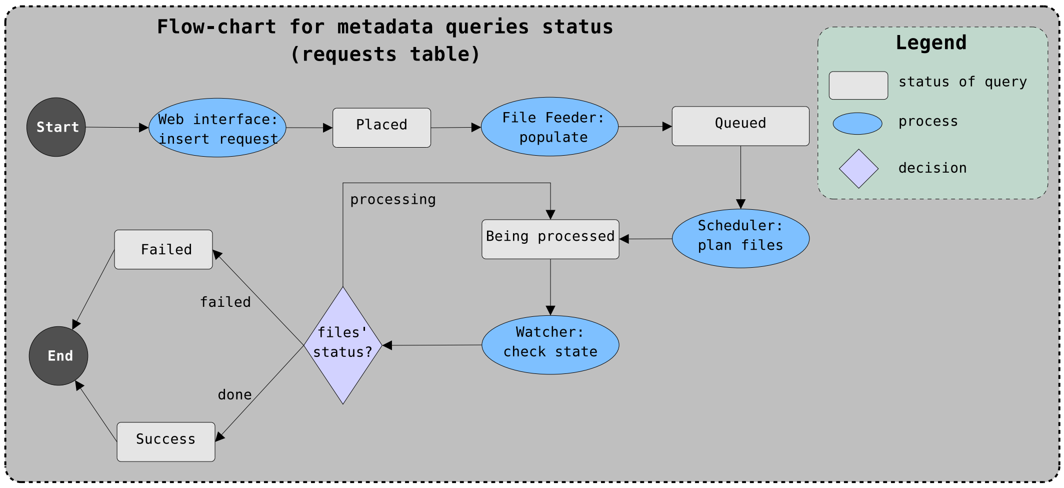


Figure 4.9: Flowchart representing the status flow related to the top-level metadata queries in requests table.

Updating and deleting accounting and monitoring tables

Accounting table holds statistical information for calculation of links speed. During the update, we store the actual speed and QOS for each link together with the time point when the measurement was done. Since some transfer tools work with batches, estimating the speed is a bit tricky. The updating hook loops through a set of intervals and queries the **scheduled_transfers** table for files which were newly transferred (since the last update) and the transfer took less time than the current interval. The speed is then calculated as an average of all measurements through the intervals and inserted into the **accounting** table.

The **monitoring** tables keep information for statistics, which allows the web interface to create plots displaying speed and data profiles. The purpose is to see what was the amount of records in time assigned to each link by state (Placed, Queued, Being processed, Done, Failed).

Deleting accounting table is performed in order to keep the table small - as we will explain in the following paragraph, for updating the **links** table we need only several most recent values. Deleting records from the monitoring tables depends only on decision how long historical records we want to keep. All of this is defined in a configuration file.

Updating links table

Updating the links table is important to have accurate and recent speed and QOS data for each link, so the Planner can produce realistic plans. The table is updated by using N

most recent records and by computing the weighted average of them. The weight of each record represents the importance of that particular speed or QOS value and it usually decreases with the age of the record. The weights and number of them is configurable. The calculation using the Python lambda mechanism is then following:

```
speed = reduce(lambda x, y: x + y, [speeds[i] * self.accWeights[i]
                                    for i in range(len(speeds))])
```

where `speeds` is an array of most recent N speed values and `accWeights` array of particular weights. The computation of QOS is similar.

Updating requested_files table

When individual file transfers are done, the propagation to the upper **requested_files** table is not immediate. Watcher periodically checks which files are already at the destinations, which transfers have failed; and updates the higher level tables. The update involves refreshing the total number of succeeded and failed files in a request, handling the retries (if maximum not reached) and switching statuses as we described in Section 4.1.2.

Cache management

The cache space is also handled by the Watcher and this hook is available for every site participating in the system. This hook is a placeholder for cache management algorithm, as we described in Section 2.5.

4.1.4 Planner

The *Planner* (Fig. 4.10-left), the brain of the system, is built on the constraint-based mathematical model. The theoretical background and our continuous progress were described in previous sections. The solver uses methods from Constraint Programming and Mixed Integer Programming and the logic tries to minimize the makespan considering all possible combinations. The tree of possibilities may very well contain solutions where transferring data once on a given link lead to a minimum or balancing between services lead to the fastest transfers. In all cases, the optimal solution will only be determined by the input parameters. Our planning is also incremental - we have previously demonstrated ([69]) that a full plan comparing to incremental planning would not make a large difference on the makespan overall - the gain of an incremental approach is the ability to self-adapt based on the *Mover's* feedback.

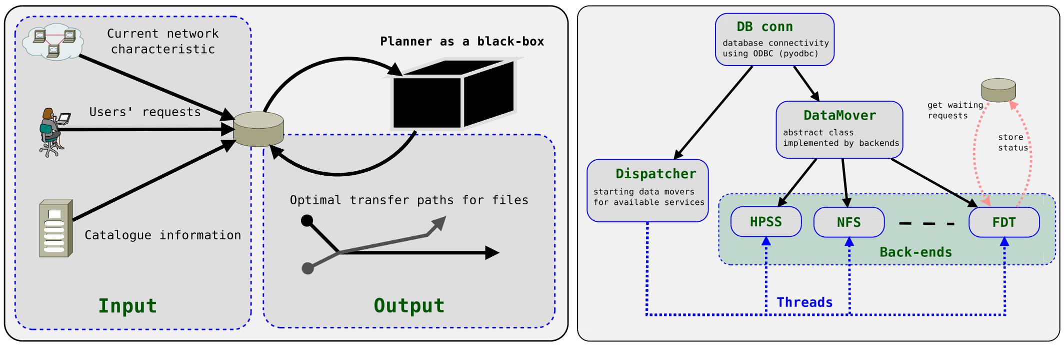

Current network characteristic
Users' requests
Catalogue information
Input
Planner as a black-box
Optimal transfer paths for files
Output
DB conn
database connectivity using ODBC (pyodbc)
DataMover
abstract class implemented by backends
get waiting requests
store status
Dispatcher
starting data movers for available services
HPSS
NFS
FDT
Back-ends
Threads

Figure 4.10: **Left:** Planner as a black box. **Right:** Data Mover component.

The Planner is written in *Java SE 6* [7] programming language. The database-independent connectivity between the Java programming language and the SQL database is handled using Java Database Connectivity (JDBC). The Planner is invoked from the wrapper script centrally from BNL site. As we remarked in theoretical sections 3.1 and 3.2, for mathematical computations Planner relies upon two libraries. **G**NU **L**inear **P**rogramming **K**it (GLPK [23]), an ANSI C library intended for solving large-scale linear programming, mixed integer programming, and other related problems is called via SWIG interface ([24]). For implementation of the CP model we use **Choco** [8], a Java based library for constraint programming. The main logic consists of gathering required data from the database and transforming them into the format understood by the implemented model in a particular library. The solution, the plan, is then transformed again back from the MIP/CSP language into the database. Such a plan then serves as the work instructions for Data Mover component.

4.1.5 Data Mover

The *Data Mover* is the distributed component responsible for performing data transfers in a reactive way. Each instance is controlling data services within a given computing site and also the wide-area network connections from/to the site. It relies on the underlying data transfer tools and uses them for data movement. In our implementation, we did not address interoperability of Wide Area Network (WAN) data transfer tools (which is not the object of this work) but settled in using by the **F**ast **D**ata **T**ransfer tool (FDT [19]). The way data movers operate is reactive which means that, as soon as a file appears at the source node (either at a data service or in a cache space before WAN transfer) it is

[7]Java: http://www.java.com
[8]Choco: http://choco.sourceforge.net

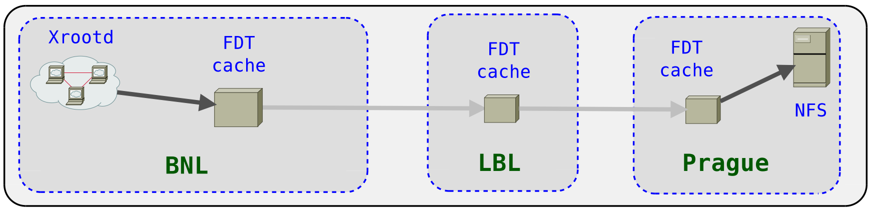


Figure 4.11: Illustration of data flow from Xrootd service @ BNL to NFS service @ Prague via intermediate cache @ LBL.

marked as “ready for transfer” and moved by the proper underlying tool. As soon as the transfer is finished another instance realizes the file is available and initiates the next move (along the computed path from the solver).

We will describe this process in more detail using an example depicted in Fig.4.11. In this example, let us suppose the transfer path for some file was advised by the Planner starting from Xrootd service (BNL site), using an intermediate cache space at LBL site and finally ending at NFS service at Prague site. There are three independent Data Movers running at each site. First, the Data Mover at BNL realizes there is a work for its thread which is responsible for Xrootd service. The transfer from Xrootd to local cache is initiated. As soon as the file is prepared and checked, the status is updated and Data Mover running at LBL side can start. In this case, the WAN transfer is started in a pull mode, and file is brought from BNL’s cache to the intermediate LBL’s one. The status is updated again and following the same principle, WAN transfer initiated by the Prague’s Data Mover may start. Finally, another thread responsible for NFS service moves the file to the requested NFS destination.

Our approach is also adaptive: from the initial transfer and consequent monitoring, the real speed can be inferred and re-injected as a parameter for the next incremental plan, helping the system to converge toward realistic transfer rates rather than relying on theoretical optimum alone.

The *Data Mover* is written in *Python* language; communication with the SQL database is handled by pyodbc [9] library (module that allows to use ODBC to connect to almost any database) and concurrent link/service control is achieved by separate threads (Fig. 4.10-right). Every module has own configuration prescribing how the underlying tool should be used, what is the number of retries in case of failures, time limit defined for single execution, etc. We will outline the major underlying tools Data Carousel relies on, namely FDT for WAN transfers, DataCarousel for HPSS read-access, Xrootd for

[9]pyodbc: http://pyodbc.sourceforge.net/

read-access from Scalla system and traditional NFS.

Fast Data Transfer (FDT)

The system is using FDT tool for WAN transfers. FDT is a client/server application capable of reading and writing at disk speed over wide area networks (with standard TCP). It is written in Java SE 6, runs an all major platforms. It is based on an asynchronous, flexible multi-threaded system and is using the capabilities of the Java NIO libraries (input/output API for working with channel and buffers). Its main features are:

- Streams a dataset (list of files) continuously, using a managed pool of buffers through one or more TCP sockets.
- Uses independent threads to read and write on each physical device.
- Transfers data in parallel on multiple TCP streams, when necessary.
- Uses appropriate-sized buffers for disk I/O and for the network.
- Restores the files from buffers asynchronously.
- Resumes a file transfer session without loss, when needed

The FDT has been tested and used in the STAR collaboration for several years. The Data Mover invokes FDT client in a pull mode with a file list containing the files which needs to be transferred (pulled) from the opposite FDT server. The number of files composing a filelist is configurable. Setting this number adjusts the atomicity of the single WAN transfer.

DataCarousel

The interaction (reading) with the Mass Storage System (HPSS) is handled by the fault tolerant policy driven framework called DataCarousel [35] (already outlined in Section 1.3.2). The tool has been developed in STAR and is in use since 2001. It is based on client/server mechanism and allows requests for archived files to be managed and coordinated the same way as “full-fledge” batch system would. It is entirely written in Perl [10] and uses SQL based database storage as the back end. The client is a thin script which sole purpose it to add requests to a central database. The server is the heart of the system - it sorts the records, creates a job of files to retrieve and submits that job to HPSS (software at another layer) according to policies.

[10]Perl: http://www.perl.org/

The integration into the Data Mover is similar to the FDT. The client is invoked with a list of files that are requested (planned) to stage from HPSS. The transfer mode is asynchronous - the client just enters the request into the DataCarousel's database and returns. Data Mover itself waits and checks for the presence of requested files (considering time limit). The number of files that are submitted to DataCarousel in a batch is configurable. It is important to remark that this number influences the performance of the full system. If the number is too large, once the HPSS slows down or other service increases its performance, the system has to wait longer for the files to be staged - instead of using some other service if available. On the other hand if the number of files is too small, the performance of the HPSS is not optimal as was shown in [30]. One can see there are multiple factors biasing the overall motion of the system.

Scalla/Xrootd and NFS

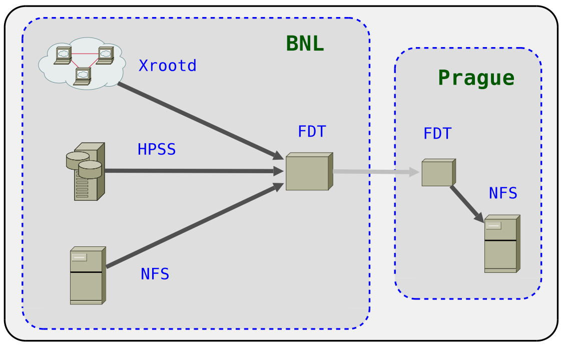
We have presented the overview of the Scalla/Xrootd system already in Section 1.3.2. It aggregates data spread over hundreds of disks and provides them to the clients running mostly as jobs written with the use of ROOT [11] analysis framework. Along with this Scalla/Xrootd provides the command-line client called *xrdcp* for retrieving files from the cluster to the local disk. Data Mover uses *xrdcp* in a synchronous mode for getting files from the Scalla system.

The communication with the traditional NFS system is handled similarly in a synchronous mode using standard tools provided by the operating systems.

4.1.6 Show case

To prove the validity and soundness of our planning strategy in practice, one has to design and implement several cases. The system has to be tested and observed under different and changing condition to see if it reacts and works as expected. We will start with the simple short-time test to affirm the software components work and communicate in the expected way and the quality of the computed plan is confident. We can look at the environment for this case as "ideal", where all data services and network were working smoothly without any breakdowns. The environment was for simplicity formed by two computing sites, the central **BNL** and remote **Prague**. The available data services at *BNL* were: *Xrootd*, *NFS* and *HPSS*, while in *Prague* only *NFS* was available. The wide area network (WAN) transfer was controlled by FDT. The configuration is shown in Fig. 4.12-left. The purpose was hence to challenge the planner in making proper decisions when

[11]ROOT: http://root.cern.ch



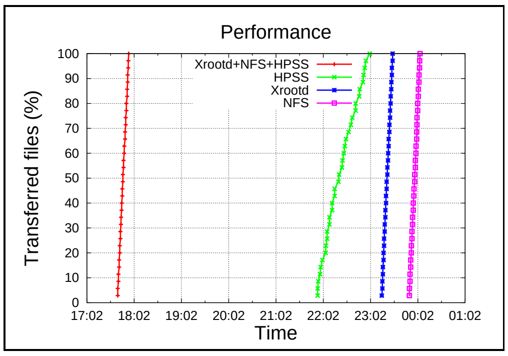


Figure 4.12: **Left:** The network and service configuration for the tests. **Right:** The performance of the system using 4 different configurations. On the X axis, we represent the time of transfers while Y is the percentage completion. The x-range of each curve is hence representative of the makespan.

multiple sources were available at the same site. We were interested to see how fast the requested files can be brought to the destination if we restrict the system to reason only about particular sources.

The request consisted of files available at all data services at *BNL* at the same time and the task was to bring them to the *Prague* NFS service. The test was composed of four different configurations. The planner consecutively considered:

- only Xrootd repository
- only NFS repository
- only HPSS repository
- a combination of Xrootd, NFS and HPSS repository concurrently

The results of each configuration are shown in Fig. 4.12-right. As expected, while all files are located on mass storage in STAR, transfers from *HPSS* (in green) are the longest to accomplish and hence, lead to the longest delays in delivery. In our setup, the green and blue curves are near equivalent (*NFS* direct transfers are slightly faster) but it is to be noted that not all files are held on *NFS* (central storage) in STAR and pulling all files from *Xrootd* may cause significant load on a system in use primarily for batch based user analysis (hence, an additional load is not desirable). When we combined all storage sources, the makespan was equivalent to the one from *Xrootd* while the relative ratio of files transfers from the diverse sources was 19%, 38% and 43% for *HPSS*, *NFS* and *Xrootd* respectively with no load caused on any of the services. At the end, the overall bottleneck was only the WAN transfer speed - we infer our test proved the planner

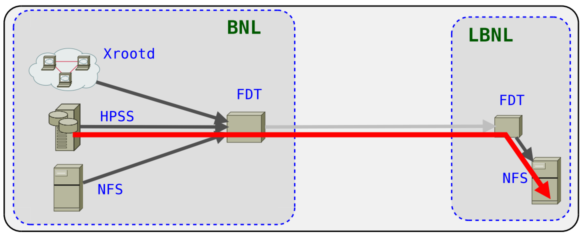


Figure 4.13: Direct data flow.

works as expected, since the full reasoning considering all possible repositories led to the optimum makespan. Additionally, the utilization of all services brings the advantage in the form of load-balancing and automatic use of replicas.

The test above was facing ideal conditions where none of the services was fluctuating in a performance and the test was short termed. To validate the workflow between the software components (explained in Section 4.1.2) and to see the proper file status propagation we present another real-case.

The task was to bring dataset from BNL's HPSS to LBNL's NFS storage. In this case, there is a straight flow of data, since files were not yet replicated (see Fig. 4.13). This allows us to concentrate on data and information passing mechanism between Data Movers and inspect the monitoring interface.

The dataset containing more than 3100 files was defined by the following catalogue request:

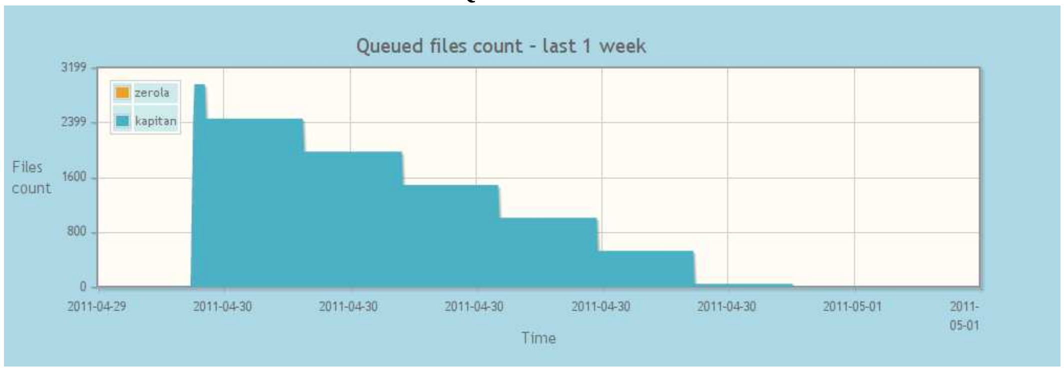
- **trgsetupname**=production_dAu2008
- **filetype**=online_daq, **filename** $\approx$st_zerobias
- **tpc**=1,**tpx**=1,**emc**=1,**runnumber**[]8341061-9027091,**events**$>$20

In this setup, there were two Data Movers. One running at BNL's site responsible for acquiring files from HPSS to local cache, and another one running at LBNL's site and performing WAN transfers and consequent NFS placements. We will first inspect how the system cooperates with underlying HPSS service using DataCarousel tool. Graphs in Figures 4.14 and 4.15 represent the amount of files with status "Queued" or "Being processed" assigned to the particular service as a function of time. Let us look first at central BNL site.

The system schedules all files to be acquired from HPSS, since it was the only repository holding the dataset. The BNL's Data Mover was submitting files to DataCarousel in batches of 500. As soon as the batch arrived (the status of submitted files changed from "Being processed" to "Done" or to "Failed") the system took another burst of queued files

HPSS performance

Queued files



Processing files

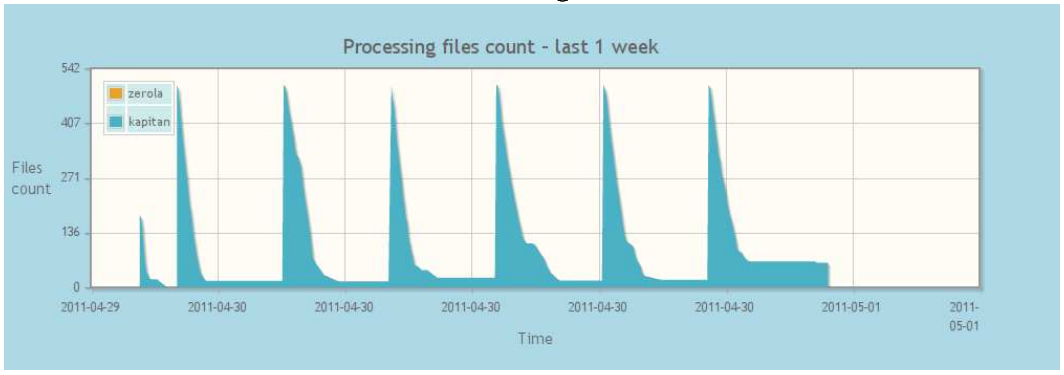
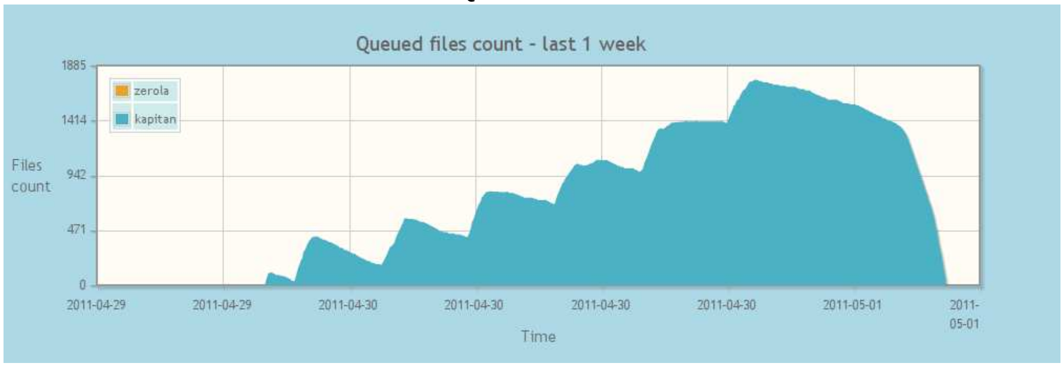


Figure 4.14: **Top.** Graph displaying the number of files with status “queued” assigned to the HPSS Data Mover as a function of time. **Bottom.** Identical plot inspecting files with status “Being processed”.

FDT performance

Queued files



Processing files

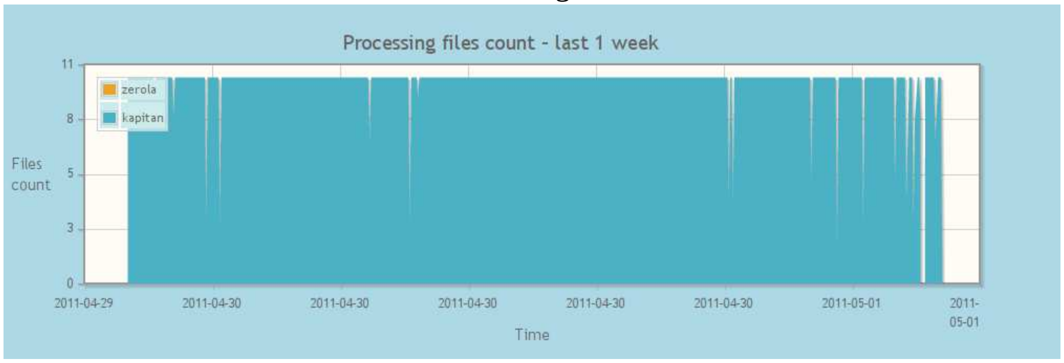


Figure 4.15: **Top.** Graph displaying the number of files with status "queued" assigned to the FDT WAN Data Mover as a function of time. **Bottom.** Identical plot inspecting files with status "Being processed".

and submitted them too. This can be seen in Fig.4.14-bottom as spikes in a graph, where each spike is defined by the speed how fast the files in a current batch were staged. Similarly, the amount of queued files was decreasing, what is plotted as steps in Fig.4.14-top. Let us move up now to the destination LBNL site.

As soon as first batches of files were staged from HPSS and prepared for WAN transfer, the number of queued files for FDT service (initiated in pull mode from LBNL) started to increase (Fig.4.15). We can see that for the whole time of movement the WAN was saturated - the number of processing files was constant. This implies that the DataCarousel was able to prepare files from HPSS tapes faster than WAN allowed to transfer. Hence, we see the increasing number of queued files, waiting for FDT service.

This data movement took approximately 1 day and at the end system reported 108 files that failed to be staged, even upon retries.

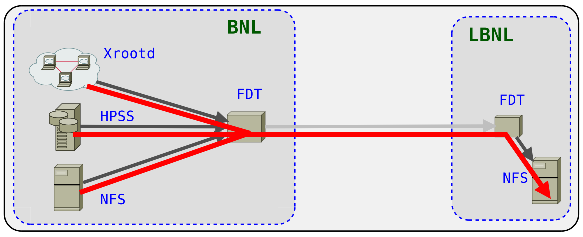


Figure 4.16: Data transfer scheme for P10ik dataset.

Adaptation to service failure

It is important to verify whether the system reacts in case of service failures the way it is expected to act. We will present our experience from the following task. The system has been used actively for replicating the production dataset *P*10*ik* from STAR's *Tier-0* center BNL to LBNL (*Tier-1* center). The size of the dataset was about 48TB and at the time of writing the text more than 10TB has been already transferred. The corresponding catalogue request was:

- **trgsetupname**=AuAu39_production
- **production**=P10ik
- **filetype**=daq_reco_MuDst, **filename** ≈st_zerobias

The system was leveraging all data sources from BNL site, as can be seen at Fig. 4.16. We will focus now on the situation how the system handles underlying service failures. As we can see, last part of the data transfer chain is using NFS service for storing data at the final destination. In the case of high access loads or problems of servers exporting the mounts, there may appear temporary hang-ups. During this time, the *Data Mover* instance responsible for NFS service is waiting and number of queued files at NFS service is increasing. This happened also during the move of *P*10*ik* dataset as we can see in Fig. 4.17. The graph is showing the increase trend of files the *Data Mover* received but was not able to store due to the stalled NFS service. What we want to illustrate now is that the system realized the problem and the appearing bottleneck at the end of the transfer chain. The system realized that due to the failed NFS service there was an increasing number of queued files at the last part of transfer chain and adapted to this situation in further planning (see Fig.4.18). Since there was no reason to increase even more the number of stucked files, the system reacted accordingly and postponed the transfer of further files. This can be seen in Fig. 4.18 as a part of graph with decreasing tendency of queued files at FDT *Data Mover*. As soon as the problem was resolved the

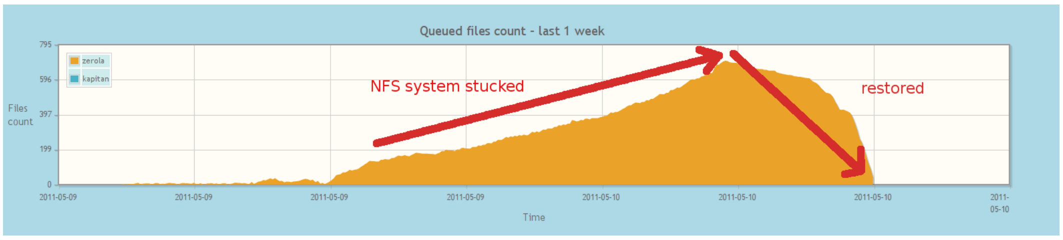


Figure 4.17: Graph displaying the number of files with status "queued" assigned to the NFS Data Mover as a function of time. Because of not responding NFS server, the number of queued files started to increase until the server got back to the normal mode.

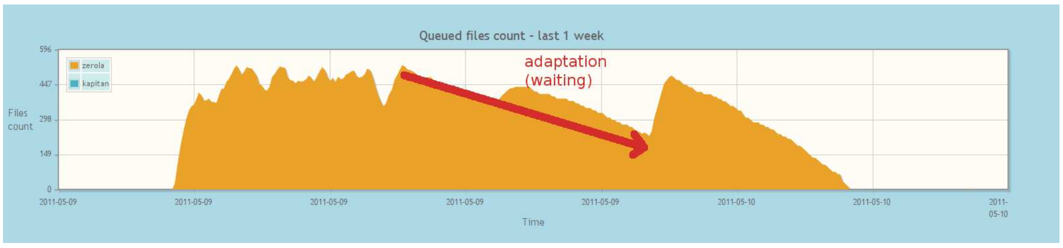


Figure 4.18: Graph displaying the number of files with status "queued" assigned to the FDT Data Mover as a function of time. The system realized the increasing bottleneck due to failed NFS service and adapted the further plans.

system started to plan files again in regular mode and number of queued files retained back to the normal level.

4.1.7 Performance comparison

In this section we will elaborate on the performance of the automated system under different environment configurations. The principal benefits of the solver depend on leveraging available data services and network links between sites. Let us focus first on performance comparison experienced by relying on data sources with diverse characteristic. All the following measurements were taken in real production environment while monitoring other exterior activities and requests for shared system resources. The monitoring included the control of *HPSS* usage over submitting large-size requests, extensive *WAN* third-party transfers between laboratories, etc. Therefore, the measurements spanning over several hours (and often repeated multiple times) are statistically stable and sound.

The performance of the system can be nicely described by comparing the makespan - the time it takes to bring requested files to the destination. It is important to see also the convergence, how fast were files appearing at the destination. Hence, we will display this tendency as a function of time in the following graphs. Figure 4.19 is displaying the comparison when the system alternates reasoning about data sources. The transfer was required between STAR *Tier-0* BNL laboratory and *Tier-1* center at LBNL, without involving any third site. We could concentrate thereby entirely on data sources and eliminate the influence of diverse network paths. We were comparing two data services which served as a source of the data. The services are in contrast by different characteristic:

- **HPSS** - holds all the files, works asynchronously. Usually involves waiting time at the beginning upon submission, then high throughput.

- **Xrootd** - holds only the portion of data, works synchronously. Usually provides low latency and high throughput.

The comparison consists of three several hours long transfers when solver was acting always in different mode. The blue line represents the mode when solver was using exclusively only *Xrootd* as the source service. We can see that files started to appear almost immediately at the destination and the slope of the line shows the fastest throughput. For better resolution the small plot in the same figure is displaying the zoomed region in the first 40 minutes. However, since *Xrootd* repository holds only a portion of all data, full data set could not be transferred. What portion of data the service holds usually depend on the age of files. Files from recent datasets are more likely to be available at *Xrootd* service. The red line represents the mode when system was relying only on HPSS. We can see that there was an initial small waiting time until files started to appear. More important is to realize also the “step-like” trend of the line. This is caused by the HPSS utilization. HPSS prioritizes requests from different users depending on tape locations, history, and other factors; and when there is our requests in turn, it serves the files usually fast. During several hours our request can be postponed and others are prioritized and we have to wait. Unfortunatelly, this “step” can often take several hours, depending on the current circumstances and load. The third black line is representing the last mode, when the system relied on both services concurrently. We can clearly see that the combination of both sources outperforms counting on the stable but not the fastest one (*HPSS*) even if *Xrootd* cannot provide all the files. Realizing which files where reside and coordinating the access to providing services is not something what users can efficiently do by themselves and hence, they often rely on the single one - stable but slow one. This is when automated solver can bring a significant benefit and increase the effectivity of their work.

Let us bring our attention now to the system’s reasoning and utilizing diverse network paths. For the purpose of keeping the environment transparent and eliminating the effect

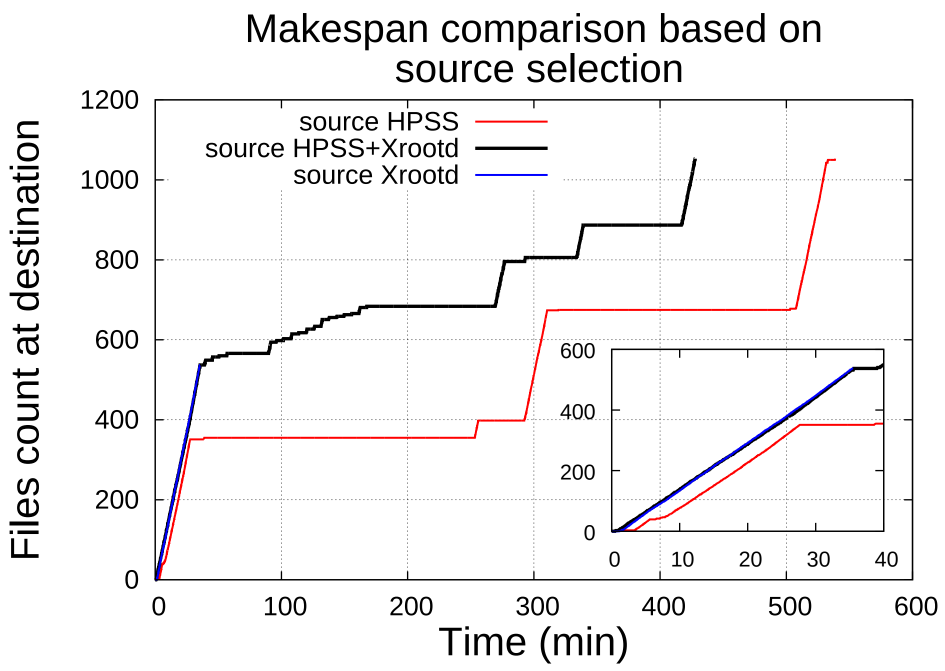


Figure 4.19: Graph displaying the trend how fast is the plan fulfilled concentrating on 3 different modes of source access. First one uses solely *Xrootd* service, second only the *HPSS* system, and finally third one uses combination of both.

of multiple data services, in this case we will concentrate on the sole *Xrootd* service as the source of all files. The slow latency and constant modest bandwidth of this data service allow us to concentrate on the influence of reasoning about network paths. The data transfer scheme is illustrated in Fig.4.20, where data are being moved from BNL laboratory (Tier-0 center) to Prague site (Tier-2 center). The system is allowed to reason also about intermediate site (LBNL laboratory, Tier-1 center) in order to increase the throughput. It is important to state that the connection between the BNL and Prague site is using static routing over dedicated link and is diverse from the path between BNL and

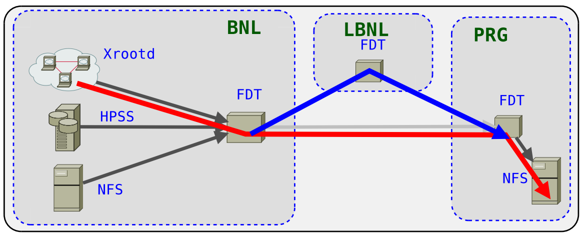


Figure 4.20: Transfer scheme for diverse network paths. Data movement relies on the *Xrootd* service as the sole source of files while leveraging also intermediate LBNL site.

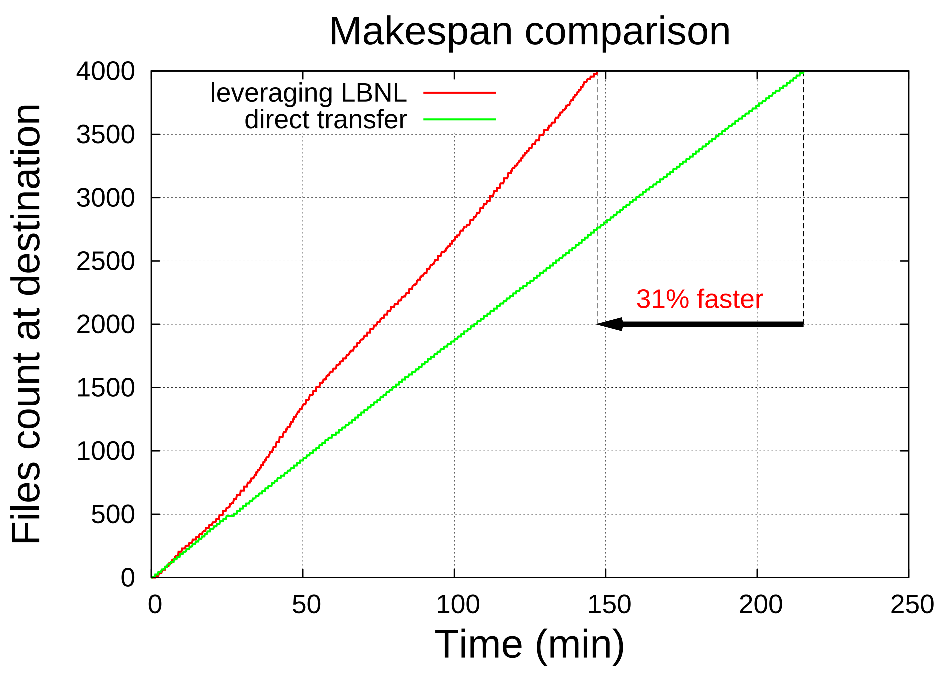


Figure 4.21: Graph displaying the trend how fast is the plan fulfilled comparing 2 different modes in network paths. Green line represents the mode when only direct transfer to Prague was allowed and red one the mode when part of the traffic was allowed to be routed via LBNL site.

LBNL as well as LBNL and Prague (using ESnet, Geant, and CESNET routing). We will again compare the speed how files appear at the destination service while looking at the impact of using intermediate site in parallel.

The graph in Fig.4.21 is exposing this comparison. The green line represents the mode where only direct BNL to Prague network path was used; while the red line the mode where also additional path through LBNL site was allowed. We can see the clear and very visible benefit in leveraging additional network path and routing part of the traffic via intermediate site. The overall gain in makespan (how less user waited for files) was almost one third of total time comparing to the direct and usual approach.

Chapter 5

Conclusions and future work

This work deals and attacks the complex problem of efficient data movements on the network within a distributed environment. Although the work has included theoretical research it was shaped from the beginning in order to deliver practically useful tools. Therefore it is willing to provide balanced combination of theoretical and practical results.

The problem itself arises from the real-life needs of the running nuclear physics experiments, while all the studies were obtained on the running experiments with more than a decade long history - experiment STAR, and its peta-scale requirements for data storage and computational power. Unlike projects which rely purely on simulations we have been working with real STAR data, infrastructure and services during regular operation of the experiment.

This has been the first time in nuclear physics large scale experiments when automated planning approach was used for reasoning about data transfers and cpu allocations. We showed that computational complexity of the transfer problem is strongly NP-hard using polynomial-time reduction from $J3|p_{ij} = 1|C_{\max}$, a 3-machine unit-time job-shop scheduling problem (JSSP).

We proposed and presented the two stage constraint model, coupling path planning and transfer scheduling phase for data transfers to the single destination. Several techniques for pruning the search space, like symmetry breaking, domain filtering, or implied constraints, are shown. We proposed and implemented several search heuristics for both stages and performed experiments for evaluating their applicability.

Further, we presented the extension of the solver which was based on pure Constraint Programming techniques for multi-site data movement within the multi-user environment. We introduced the Mixed Integer Programming methods into the model and several simplifications in scheduling phase. The comparison proved that the simplification of the scheduling phase and linearization of the constraints lead to the faster solving times while loosing only neglecting fraction of the quality of the makespan.

We have implemented the model and designed the architecture, components and performed implementation of the framework with other STAR services. Simplistic yet robust architecture allows users to express their requests conveniently via a Web interface while the back-end planner and the set of data movers take care of the work on the user's behalf. We have presented observations and results on how the system behaves from multiple long term measurements that were taken in STAR. We focused on service failure recovery, comparison of overall makespan improvement against classic techniques and benefits of automated leveraging multiple data sources and diverse network paths.

With upcoming requirements for more frequent Cloud computing where data storage is constrained and the needs for prompt feeding CPUs by data is important, the automated data transfers and job allocation can greatly simplify the user's task. We have addressed this and proposed the extension of the model for reasoning about computational power as well.

It is always positive when solving approach and technique is universal enough so it is possible to apply it in other related areas. We have researched also the field of robotics and automated planning used in autonomous robots. Derived techniques from planning the data transfers in large scale physics experiments were successfully applied in vehicle routing variants.

The journey to the fully automated and intelligent system scheduling user's tasks considering all relevant constraints is certainly long and there are still numerous aspects that need to be addressed. The intelligent cache management reflecting the prediction of the future needs or advanced bandwidth reservation might be the ideas for the next improvements of the system. We believe that the presented work contributed to this collaborated effort with several sound ideas, techniques and concepts.

List of Figures

Appendix A

Towards Routing for Autonomous Robots

Path planning is one of the critical tasks for autonomous robots. We will study the problem of finding the shortest path for a robot collecting waste spread over the area such that the robot has a limited capacity and hence during the route it must periodically visit depots/collectors to empty the collected waste. This is a variant of often overlooked vehicle routing problem with satellite facilities. We present two approaches for this optimisation problem both based on Constraint Programming techniques. The former one is inspired by the operations research model, namely by the network flows, while the second one is driven by the concept of finite state automaton. The experimental comparison and enhancements of both models are discussed with emphasis on the further adaptation to the real world environment.

A.1 Introduction

Recent advances in robotics have allowed robots to operate in cluttered and complex spaces. However, to efficiently handle the full complexity of the real-world tasks, new deliberative planning strategies are required. We deal with the robot performing a routine task of collecting waste for example in large department stores where the remote control is boring for humans and hence error prone. In particular, we solve the problem of planning a route for a single robot such that all waste is collected, robot's capacity is never exceeded, and the route is as short as possible. We assume the environment to be known and not changing, in particular, the location of waste and depots is known and the robot knows how to move between these locations. To handle changes in the environment we

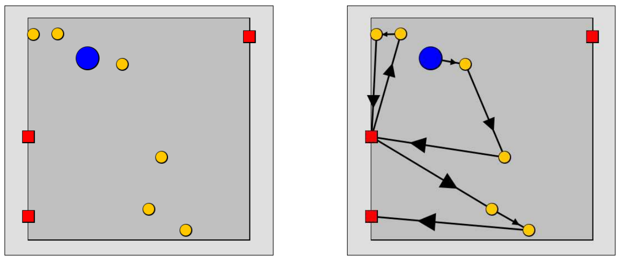


Figure A.1: Example of 6+3 robot planning task. The robot (the big circle) collects waste (six small circles) and uses collectors (three squares) to empty the bin when it is full.

focus on anytime planning algorithms that can be re-run when the initial task changes, for example, the distances between the navigation points change due to cluttered areas. We propose to use Constraint Programming (CP) to solve the problem because of the flexibility of CP. This allows us to use a base model describing the core task and to add new constraints later when necessary. Such a new constraint could be the restriction on allowed combinations of entrance and exit routes when collecting the waste or visiting the depot for robots with limited manoeuvring capabilities. Figure A.1 gives an example of the initial environment (left) and the found path for the robot (right). The task we are dealing with is to develop a robot solving a specific routing problem - an often overlooked variant of the standard Vehicle Routing Problem (VRP). In our setting, the robot has to clean out a collection of waste spread in a building, but under the condition of not exceeding its internal storage capacity at any time. The storage tank can be emptied in one of available collectors. The goal is to come up with the routing plan minimising the travelled trajectory. This is a similar setting to a Vehicle Routing Problem with Satellite Facilities (VRPSF) studied in (Bard et al. [4]), where the task is to deliver goods rather than to collect waste. Our primary goal is to develop an algorithm that returns good solutions in a short time (almost anytime algorithm) and that can be easily extended by additional constraints. Hence ad-hoc exact techniques are not appropriate due to long runtime and limited extendibility and we decided to use Constraint Programming to solve the problem. Neither of existing CP-oriented works solves the above problem, but we can use them as the initial motivation for the design of our constraint model. Most of the routing models are based on the formulation of the problem using network flows (Simonis [56]) so we also proposed a constraint model based on this standard technique. Nevertheless, the performance of this model was not satisfactory in our experiments so we proposed a radically new approach to model the problem using a finite state automaton. In our preliminary experiments, this model outperforms the traditional model and can solve larger instances of the problem. The text is organised as follows. We will first formally de-

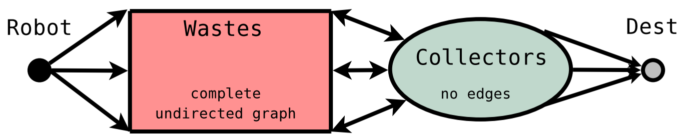


Figure A.2: A schema of the graph describing the robot's environment with the navigation points.

scribe the problem to be solved. Then we will formulate the traditional model based on network flows that we customised to solve our problem. After that we will describe the novel model based on finite state automata. Finally, we will conclude the preliminary experimental results.

A.2 Problem formulation

Recall that we are solving a single robot path planning problem with the capacity constraint. The robot's environment consists of the navigation points defined by the locations of waste and collectors. We use a mixed weighted graph (V,E) with both directed and undirected edges to represent this environment. The reason for using undirected edges is minimising the size of the representation. The set of vertices $V = \{I\} \cup W \cup C \cup \{D\}$ consists of the initial position I, the set W of waste vertices, the set C of collectors and the destination vertex D. From the initial position the robot has to visit some waste so we have directed arcs from I to all vertices in W. The robot can travel between the waste vertices so we assume a complete undirected graph between vertices in W. From any waste vertex the robot can go to a collector so we use a directed edge there and from any collector we can go to any waste which is again modelled using a directed edge. We need directed edges here as we need to count the number of incoming and ongoing edges for the collectors. There are no edges between the collector vertices. As mentioned, we use a dummy destination vertex that is connected to all collector vertices by a directed edge. The weight of each edge describes the distance between the navigation points. The edges going to the dummy destination vertex D has zero weight so the robot can actually finish at any collector. The task to find a minimal-cost path starting at I, finishing at D and visiting each vertex in W exactly once such that the number of any consecutive vertices from W does not exceed the given capacity of the robot. Figure A.2 shows the schema of the graph with the navigation points.

A.3 CP model based on network flows

The first model that we propose resembles the traditional operations research models of vehicle routing problems based on network flows and Kirchhoff's laws. Basically, we are describing whether or not the robot traverses a given edge. For every edge e we introduce a binary decision variable X_e stating whether the edge is used in the path (value 1) or not (value 0). Let $IN(v)$ and $OUT(v)$ denote the set of incoming and outgoing directed edges for the vertex v. For example, for $v \in W$ the set $IN(v)$ contains the arc from the vertex I and the arcs from the vertices in C. Let $ICD(v)$ be a set of undirected edges incident to vertex v. This set is empty for the collector vertices; for waste vertices it contains undirected edges connecting the vertex with other waste vertices. The following constraints describe that the robot leaves the initial position I, reaches the destination position D, and enters each collector c the same number of times as it leaves it:

$$\sum_{e \in \mathbf{OUT}(I)} X_e = 1, \quad \sum_{e \in \mathbf{IN}(D)} X_e = 1, \tag{A.1}$$

$$\forall c \in \mathbf{C} : \sum_{e \in \mathbf{OUT}(c)} X_e = \sum_{e \in \mathbf{IN}(c)} X_e \tag{A.2}$$

Let us now describe the constraint that each waste vertex w is visited exactly once. It means that exactly two edges incident to a waste vertex w are active (used in the solution path) and there can be at most one active incoming and outgoing directed edge connecting the waste with the collectors or with the initial node.

$$\forall w \in W : \sum_{e \in \mathbf{OUT}(w) \cup \mathbf{IN}(w) \cup \mathbf{ICD}(w)} X_e = 2, \tag{A.3}$$

$$\forall w \in W : \sum_{e \in \mathbf{OUT}(w)} X_e \leq 1, \tag{A.4}$$

$$\forall w \in W : \sum_{e \in \mathbf{IN}(w)} X_e \leq 1 \tag{A.5}$$

The above constraints describe any path leading from I to D, but they also allow isolated loops as Figure A.3 shows. This is a known issue of this type of model that is usually resolved by additional sub-tour elimination constraints forcing any two subsets of vertices to be connected. In our particular setting, we need to carefully select these pairs of subsets of vertices because there could be collector vertices that are not visited. Hence, we consider any pair of disjoint subsets $S_1, S_2 \subseteq (W \cup C)$, such that neither S_1 nor S_2 consists of collector vertices only. More precisely, we assume the pairs of subsets S_1, S_2

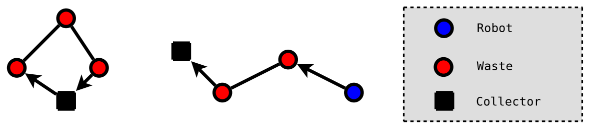


Figure A.3: An ineligible loop (left) satisfying the routing (*Kirchhoff's*) constraints.

such that:

$$S_2 = (W \cup C) \setminus S_1, \quad S_1 \cap W \neq \emptyset, \quad S_2 \cap W \neq \emptyset \tag{A.6}$$

The sub-tour elimination constraint can then be expressed using the following formula ensuring that there is at least one active edge between S_1 and S_2.

$$\sum_{e \in E : e \cap S_1 \neq \emptyset \,\wedge\, e \cap S_2 \neq \emptyset} X_e \geq 1 \tag{A.7}$$

Clearly, there is an exponential number of such pairs S_1 and S_2, which makes it impractical to introduce all such sub-tour elimination constraints. Some authors (Pop [15]) propose using single or multi-commodity flow principles to reduce the number of constraints by introducing auxiliary variables. However, our combination of directed and undirected edges makes it complicated to use this approach so we rather applied another approach based on lazy (on-demand) insertion of sub-tour elimination constraints. Briefly speaking, we start with the model without the sub-tour elimination constraints and we find a solution. If the solution forms a valid path then we are done. Otherwise we identify the isolated loops, add the sub-tour elimination constraints for them and start the solver with the updated model. This process is repeated until a valid path is found. Obviously, it is a complete procedure because in the worst case, all sub-tour elimination constraints are added.

It remains to define the constraints describing the limited capacity of the robot. For this purpose we introduce auxiliary non-decision capacity variables C_v for every waste vertex $v \in W$. These variables indicate the amount of waste in the robot after visiting the particular vertex. The non-decision character of the variables means that they are not instantiated by the search procedure, but they are instantiated by the inference procedure only. In particular, if their domain becomes empty during inference then it indicates inconsistency. The following constraints are used during the inference ($w \in W$). First, if the waste vertex w is visited directly after the collector then there is exactly one waste in the robot:

$$\sum_{e \in \mathbf{IN}(w)} X_e = 1 \implies C_w = 1 \tag{A.8}$$

Second, if the waste vertices u and v are visited directly before respectively after w (or vice versa) then the following constraints must hold between the capacity variables:

$$\forall e, f \in \mathbf{ICD}(w), e = \{u,w\}, f = \{w,v\} : X_e + X_f = 2 \implies |C_u - C_v| = 2 \tag{A.9}$$

$$\forall e = \{u,w\} \in \mathbf{ICD}(w) : |C_u - C_w| = 1 \tag{A.10}$$

Finally, to restrict the capacity of the robot by constant *cap* we use the following constraints for the capacity variables:

$$\forall w \in W : 1 \leq C_w \leq cap \tag{A.11}$$

The objective function to be minimised is the total cost of edges used in the solution path:

$$Obj = \sum_{e \in E} X_e \cdot weight(e) \tag{A.12}$$

where $weight(e)$ is the weight of edge e.

A.3.1 Search procedure

The constraint model describes how the inference is performed so the model needs to be accompanied by the search procedure that explores the possible instantiations of variables X_e. Our search strategy resembles the greedy approach for solving Travelling Salesman Problems (TSP) (Ausiello et al. [3]). The variable X_e for instantiation is selected in the following way. If the path is empty, we start at the initial position I and instantiate the variable $X_{\{I,w\}}$ such that $weight(\{I,w\})$ is the smallest among the weights of arcs going from I. By instantiating the variable we mean setting it to 1; the alternative branch is setting the variable to 0. If the path is non-empty then we try to extend it to the nearest waste. Formally, if u is the last node in the path then we select the variable $X_{\{u,w\}}$ with the smallest $weight(\{u,w\})$, where w is a waste vertex. If this is not possible (due to the capacity constraint), we go to the closest collector. The optimisation is realised by the branch-and-bound approach: after finding a solution with the total cost *Bound*, the constraint $Obj < Bound$ is posted and search continues until any solution is found. The last found solution is the optimum.

A.4 CP model based on finite state automata

The second model that we propose brings a radically new approach not seen so far when modelling VRPs or TSPs. Recall that we are looking for a path in the graph that satisfies some additional constraints. We can see this path as the word in a certain regular language. Hence, we can base the model on the existing regular constraint (Pesant [45]). This constraint allows a more global view of the problem so the hope is that it can infer more information than the previous model and hence decreases the search space to be explored.

First, it is important to realise that the exact path length is unknown in advance. Each waste vertex is visited exactly once, but the collector vertices can be visited more times and it is not clear in advance how many times. Nevertheless, it is possible to compute the upper bound on the path's length. Let us assume that the path length is measured as the number of visited vertices, the robot starts at the initial position and finishes at some collector vertex (we will use the dummy destination in a slightly different meaning here), and the weight/cost of arcs is non-negative. Let $K = |W|$ be the number of waste vertices and $cap \geq 1$ be the robot's capacity. Then the maximal path length is $2K+1$. This corresponds to visiting a collector vertex immediately after visiting a waste vertex. Recall that each waste vertex must be visited exactly once and there is no arc between the collector vertices.

Our model is based on four types of constraints. First, there is a restriction on the existence of a connection between two vertices - a *routing constraint*. This constraint describes the routing network (see Figure A.2). It roughly corresponds to the constraints A.1-A.5 from the previous model. Note that the sub-tour elimination constraints A.6-A.7 are not necessary here. Second, there is a restriction on the robot's capacity stating that there in no continuous subsequence of waste vertices whose length exceeds the given capacity - a *capacity constraint*. This constraint corresponds to the constraints A.8-A.11 from the previous model. Third, each waste must be visited exactly once, while the collectors can be visited more times (even zero times) - an *occurrence constraint*. This restriction was included in the constraints A.1-A.5 of the previous model, while we model it as a separate constraint. Finally, each arc is annotated by a weight and there is a constraint that the sum of the weights of used arcs does not exceed some limit - a *cost constraint*. This constraint is used to define the total cost of the solution as in A.12.

In the constraint model we use three types of variables. Let $N = 2K+1$ be the maximal path length. Then we have N variables $Node_i$, N variables Cap_i, and N variables $Cost_i (i = 1, \ldots, N)$ so we assume the path of maximal length. Clearly, the real path may be shorter so we introduce a dummy destination vertex that fills the rest of the path till the length N. In other words, when we reach the dummy vertex, it is not possible to leave it. This way, we can always look for the path of length N and the model gives flexibility to explore the shorter paths too.

The semantic of the variables is as follows. The variables $Node_i$ describe the path hence their domain is the set of numerical identifications of the vertices. We use positive integers $1,\ldots,K(K=|W|)$ to identify the waste vertices, $K+1,\ldots,K+L$ for the collector vertices $(L=|C|)$, and 0 for the dummy destination vertex. In summary, the initial domain of each variable $Node_i$ consists of values $0,\ldots,K+L$. Cap_i is the used capacity of the robot after leaving vertex $Node_i(Cap_1=0$ as the robot starts empty), the initial domain is $\{0,\ldots,cap\}$. $Cost_i$ is the cost of the arc used to leave the vertex $Node_i(Cost_N=0)$, the initial domain consists of non-negative numbers. Formally:

$$\forall i=1,\ldots,N(N=2K+1): \begin{array}{ccc} 0 & \leq & Node_i \leq K+L \\ 0 & \leq & Cap_i \leq cap, Cap_1 = 0 \\ 0 & \leq & Cost_i, Cost_N = 0 \end{array} \tag{A.13}$$

We will start the description of the constraints with the *occurrence constraint* saying that each waste vertex is visited exactly once. This can be modelled using the global cardinality constraint (Regin [50]) over the set $\{Node_1,\ldots,Node_N\}$. The constraint is set such that the each value from the set $\{1,\ldots,K\}$ is assigned to exactly one variable from $\{Node_1,\ldots,Node_N\}$ - each waste node is visited exactly once. The values $\{0,K+1,\ldots,K+L\}$ can be used any number of times. Formally:

$$\begin{array}{rrc} gcc(\{Node_1,\ldots,Node_N\}, & & \\ & \{v: & [1,1]\forall v=1,\ldots,K, \\ & 0: & [0,\infty], \\ & v: & [0,\infty]\forall v=K+1,\ldots,K+L\}) \end{array} \tag{A.14}$$

where $v:[\min,\max]$ means that value v is assigned to at least min and at most max variables from $\{Node_1,\ldots,Node_N\}$. The *gcc* constraint allows specifying the number of appearances of the value using another variable rather than using a fixed interval as in A.14. Let D be the variable describing the number of appearances of value 0 (identification of the dummy vertex) in the set $\{Node_1,\ldots,Node_N\}$, then we can use the following constraints instead of A.14:

$$\begin{aligned} gcc(\{Node_1,\ldots,Node_N\},& \\ \{v: & \quad [1,1]\forall v=1,\ldots,K, \\ 0: & \quad D, \\ v: & \quad [0,\infty]\forall v=K+1,\ldots,K+L\}) \end{aligned} \tag{A.15}$$

$$Node_{N-D} > 0 \tag{A.16}$$

The constraint A.16 says that $Node_{N-D}$ is not a dummy vertex; actually it is the last real vertex in the path. We can also set the upper bound for D by using the information about the minimal path length (*MinPathLength* is a constant computed in advance):

$$D \leq N - MinPathLength \tag{A.17}$$

These additional constraints A.16 and A.17 are not necessary for the problem specification but they improve inference (we use them in experiments).

The *cost constraint* can be easily described as

$$Obj = \sum_{1,\ldots,N} Cost_i \tag{A.18}$$

so we can use the constraints $Obj < Bound$ in the branch-and-bound procedure exactly the same way as in the previous model.

For the cost constraint to work properly we need to set the value of $Cost_i$ variables. Recall that $Cost_i$ is the cost/weight of the arc going from vertex $Node_i$ to vertex $Node_{i+1}$. Hence, we can connect the *Cost* variables with the Node variables when specifying the *routing constraint*. In particular, we use the ternary constraints over the variables $Node_i, Cost_i, Node_{i+1}$ $i = 1,\ldots,N-1$. This set of constraints corresponds to the idea of slide constraint (Bessiere et al. [8]). We implement the constraint between the variables $Node_i, Cost_i, Node_{i+1}$ as a ternary tabular (extensionally defined) constraint; let us call it link, where the triple (p,q,r) satisfies the constraint if there is an arc from the vertex p to the vertex r with the cost q. In other words, this table describes the original routing network with the costs extended by the dummy vertex. Formally:

$$\begin{aligned} link(p,q,r) \equiv \quad & \exists e \in E : e = (p,r), q = weight(e) \\ & \vee(q = r = 0 \wedge (p = 0 \vee p > K) \end{aligned} \tag{A.19}$$

$$\forall i = 1,\ldots,2K : link(Node_i, Cost_i, Node_{i+1}) \tag{A.20}$$

It remains to show how the *capacity constraint* is realised. Briefly speaking, we use a similar approach as for the routing constraint. The capacity constraint is realised using a set of ternary constraints over the variables $Cap_i, Node_{i+1}, Cap_{i+1}$ $i = 1, \ldots, N-1$, again exploiting the idea of slide constraint. The constraint is implemented using a tabular constraint, let us call it capa, with the following semantics. Triple (p, q, r) satisfies this constraint if and only if:

- q is an identification of a collector vertex $(q > K)$ or a dummy vertex $(q = 0)$ and $r = 0$
- q is an identification of a waste node $(0 < q \leq K)$ and $r = p + 1$.

Recall that the domain of capacity variables is $\{0, \ldots, cap\}$ so we never exceed the capacity of the robot. Formally:

$$capa(p,q,r) \equiv \begin{array}{c} (q = r = 0) \\ \vee(q > K \wedge r = 0) \\ \vee(0 < q \leq K \wedge r = p+1) \end{array} \tag{A.21}$$

$$\forall i = 1, \ldots, 2K : capa(Cap_i, Node_{i+1}, Cap_{i+1}) \tag{A.22}$$

Any solution to the above described constraint satisfaction problem defines a valid solution of our single robot path planning problem with the capacity constraint. Vice versa, any solution to the path planning problem is also a feasible solution of the specified constraint satisfaction problem. We omit the formal proof due to limited space.

A.4.1 Search procedure

Similarly to the previous model, it is important to specify the search strategy. In this second model, only the variables $Node_i$ are the decision variables - they define the search space. It is easy to realise that the inference through the routing constraints A.20 decides the values of the $Cost_i$ variables and the inference through the capacity constraints A.22 decides the values of the Cap_i variables provided that the values of all variables $Node_i$ are known.

When searching for the solution we first use a greedy approach to find the initial solution (the initial cost). This greedy algorithm instantiates the variables $Node_i$ in the order of increasing i in such a way that the arc with the smallest cost is preferred. We select the node to which the least expensive arc from the previously decided node leads. Naturally, the capacity constraint is taken into account so only the nodes such that the capacity is not exceeded are assumed. This search procedure corresponds to the search strategy of the previous model. The difference in models allows us to use a fixed variable ordering in the model based on finite automata which simplifies implementation of the search procedure. This second model also has fewer decision variables but a larger branching factor.

To find the optimal solution we use a standard branch-and-bound approach with restarts. To instantiate the $Node$ variables we use the *min-dom* heuristic for the variable selection, that is, the variable with the smallest current domain is instantiated first. We select the values in the order defined in the problem (the waste nodes are tried before the collector nodes). Exactly like in the first model after finding a solution with the total cost *Bound,* the constraint $Obj < Bound$ is posted and search continues until any solution is found. The last found solution is the optimum. Note that using the well known and widely applied min-dom heuristic for the variable selection is meaningful in this model because we have larger domains, while the same heuristic is useless for the previous model which uses binary domains.

A.5 Embedding CP models into local search

The current state of the art techniques for solving VRPs are frequently based on hybrid approaches. For example the paper (Rousseau et al. [52]) suggests using CP techniques to explore the neighbourhood within Large Neighbourhood Search. We decided to apply a similar approach with our CP models to check, if the solution quality can be improved in comparison with the pure branch-and-bound approaches presented above.

The basic elements in the neighbourhood local search are the concept of the neighbourhood of a solution and the mechanism for generating neighbourhoods. It is eminent that the performance and "success" of the local search algorithm strongly depends on the neighbourhood operator and its state space. In our case, the state corresponds to the plan - a valid path for the robot. The local search algorithm is repeatedly choosing another solution in the neighbourhood of the current solution with the goal to improve the value of the objective function. This move is realised by a so called *neighbourhood operator*. We have implemented an operator that is successfully used for solving the Travelling Salesman Problems. The operator relaxes the solution by removing an induced path of a given length and then it calls the CP solver to complete the solution. It means that we add to a given constraint model additional constraints that fix some edges (for the model based on network flows) or forbid using some edges (for the model based on finite state automata). These fixed edges correspond to the edges in the original solution that were not removed by the neighbourhood operator. The role of the CP solver is to optimally complete this partial solution by adding the missing edges. The new solution is the state to which the local search procedure moves.

As the local search repeatedly chooses a move that improves the value of the objective function (we are minimizing the value), it can get “trapped” in the first local minimum it encounters. In order to escape the local minimum, a controlled method of accepting an ascending move is required. In this paper, we examined the simplified simulated annealing.

Note finally, that as the initial solution for local search we used the first solution obtained from the pure CP model (see the description of the search procedures above).

A.6 Experimental results

In this section we will present the preliminary experimental evaluation of the presented solving techniques. As there is no standard benchmark set for the studied problem, we generated own problem instances. We used a square-sized robot arena where the positions of the waste and the initial location of the robot were uniformly distributed. The collectors were uniformly distributed along the boundaries of the arena and the weights set up as a point-to-point distance using the Euclidean metric. All the following measurements were performed on `Intel Xeon CPU@2.5GHz` with 4GB of RAM, running a Debian GNU Linux operating system.

A.6.1 Performance of the network flow model

As stated earlier, the model based on network flows corresponds to the traditional operations research approach, but we modified the model to describe specifics of our robot routing problem. The model was implemented in *Java SE 6* using *Choco*, an open-source constraint programming library. The optimisation search strategy uses the built-in branch-and-bound method, while all constraints correspond to the mathematic formulations described earlier.

Figure A.4 shows the runtime (a logarithmic scale) to obtain the optimal solution as a function of the instance size measured by the number of waste and by the number of collectors. We generated 15 instances for each problem size and the graph shows the average time the solver needs for finding and proving the optimality of the solution. The capacity of robot was 3.

As already mentioned in (Bard et al. [4]), the satellite facilities in VRP (or collectors in robotics case) heavily increase the complexity of the problem. The initial experiment

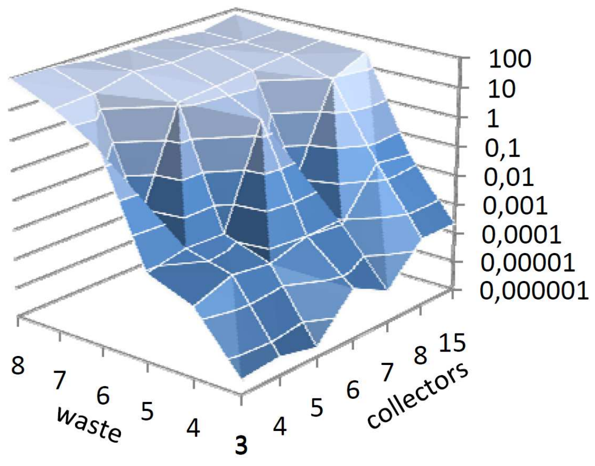


Figure A.4: Runtime (seconds) for the network flow model.

shows that the runtime increases exponentially with the number of waste but the runtime is not significantly affected by the increased number of collectors. In fact it seems that for different quantities of waste there are different numbers of collectors where the best runtime is achieved. This is an interesting observation claiming that for a given number of waste there is some number of collectors that gives the best result. Nevertheless, this observation requires additional experiments to confirm it.

While the graph in Figure A.4 represents the total time the solver needs for finding and proving the optimality of the solution, we are also interested in how fast a "good enough" solution can be found. This characteristic can be seen in Figure A.5, where the graph displays the convergence of the solution during search. We can see that even a simple greedy heuristic performs very well and the difference from the optimal solution was less than 5% within first 6 seconds for the instance $7+3$.

A.6.2 Performance of the network flow model within local search

As mentioned above, the CP model can be used within the Large Neighbourhood Search procedure to solve larger instances but obviously without any guarantee of optimality. We generated 50 independent problem instances with 20 wastes and 3 collectors (referred to as $20+3$). The capacity of the robot was set to 7 units. The neighbourhood operator was allowed to remove 5 randomly selected consecutive edges during the search and the embedded CP solver was allowed to search for 1 second. The graph in Figure A.6 shows an average one-to-one performance of the pure CP method and the LS method (with the embedded CP model) applied to the produced instances. The graph shows the difference

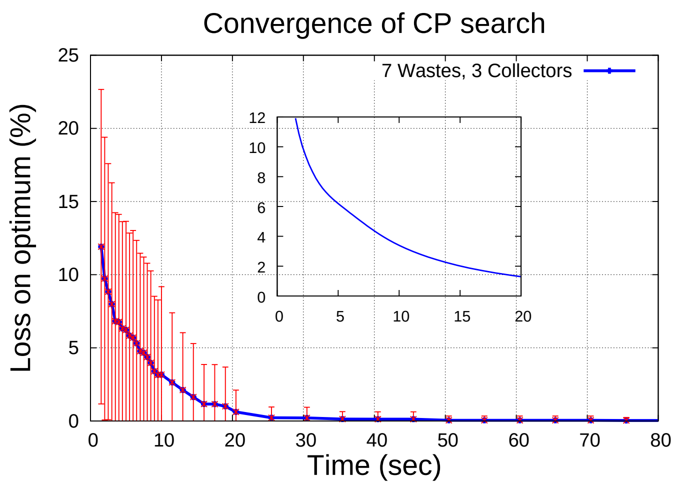


Figure A.5: Quality convergence for the network flow model.

in the quality of a solution found in the corresponding time from the LS viewpoint.

The local search procedure performed better in the long run, when compared to the pure CP method relaying only on its inner heuristic. However, CP beat LS in the first seconds where the convergence drop was steeper. As a consequence, CP seems to be a more appropriate method under very short time constraints, while reasonably good solutions can be found with a combination of LS for larger instances.

A.6.3 Performance of the finite state automaton model

The network flow model represents a standard approach to solving the Vehicle Routing Problems so we compared our novel constraint model based on the finite state automaton directly to this approach. The second model was implemented in **SICStus Prolog** [1]. Figure A.7 shows the runtime (a logarithmic scale) to obtain the optimal solution using the constraint model based on finite state automata using the same problems as for the model based on network flows (Figure A.4). The result also shows the exponential grow with the increased number of waste and weaker dependence on the number of collectors.

To directly compare both models, we generated a graph showing the difference of runtimes for the network model and for the automata model - the values above zero mean faster automata model, while the times below zero mean faster network model. Figure A.8 shows these difference times. The conclusion drawn from this graph is as follows. The automata-based model is visibly better for a smaller number of collectors where the problem is more constrained and the capacity constraints can prune more of

[1]SICStus Prolog: http://www.sics.se/sicstus

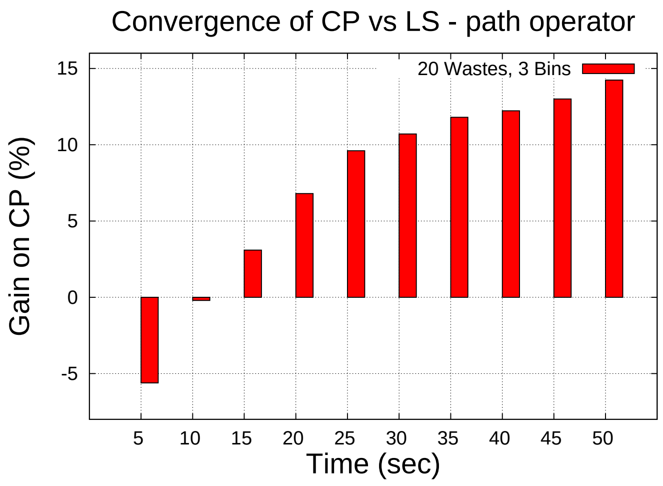


Figure A.6: Comparison of the quality convergence of the network flow model in the pure CP approach and the CP model embedded into local search.

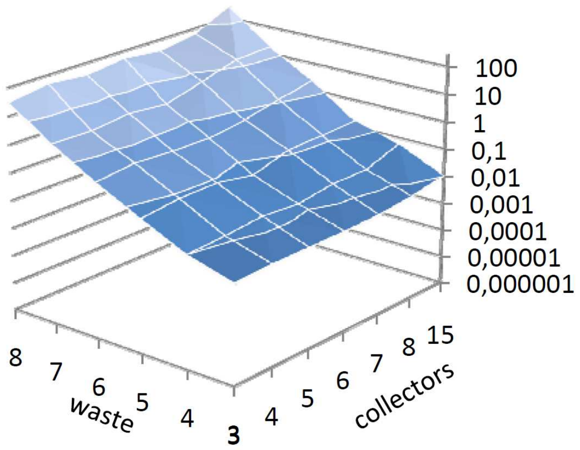


Figure A.7: Runtime (seconds) for the model based on finite state automata.

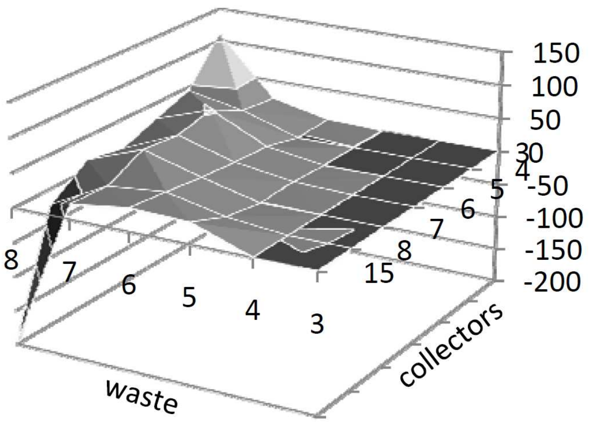


Figure A.8: Time difference (seconds) between the CP models. Positive values means that the model based on finite state automata is faster.

the search space. A bit surprisingly, it seems that the network-based model is better when the number of collectors becomes larger. This feature will require a further investigation.

Since in robotic, finding a good plan fast is more important than having the optimal one late, we started to investigate again the quality of the plans found by the CP solver in a limited time. In particular, we embedded the new CP model in the Large Neighbourhood Search procedure as described above and we tried to compare the pure CP model with this LS approach on much bigger instances with 40 wastes and 3 collectors. To our surprise, the LS method was not able to improve the solution found by the CP model in the 2 minutes runtime. As we need to produce a good solution in seconds, the pure CP model based on finite state automaton seems more appropriate for our purpose.

A.7 From high-level planning to real world

To evaluate the overall performance of the system, we used a professional commercial development environment Webots [2]. Webots is used to test robots in a physically realistic world. Our mobile robotic platform is based on the simulated *Pioneer-2* robot equipped with a laser sensor. The extreme precision of the distance measurement sensor enables reliable localization and motion planning algorithms. The part responsible for handling localization and motion planning is based on the well established open-source *CARMEN* software (Thrun et al. [60]).

[2]Webots: http://www.cyberbotics.com/

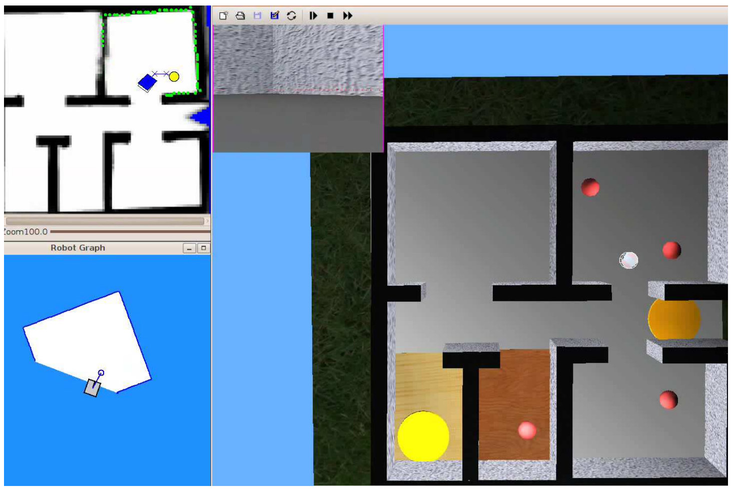


Figure A.9: Physically realistic robot simulator. **Upper left**: Map built by the robot corresponding to the testing environment with output of the motion planner module.

The very first step is to create maps (Figure A.9, upper-left) - both the input graph for the high-level path planner and the occupation grid for the collision avoidance algorithms. The individual components of the whole process are depicted in Figure A.10. The layer with the highest priority - the collision avoidance module - is activated when any obstacle is found to be too close. In that case, the robot executes an avoiding manoeuvre. The motion planner component provides the distances to the CSP planner. This is a form of integrating the traditional motion (path) planning from robotics with the more complex path planning with limited capacity to collect all waste. The found plan is delivered back to the motion planner, which produces the motion commands. The map building layer updates the map and monitors the plan execution.

In reality, the failures of individual components can cause bad performance of the overall system. For example, the failures of the localization algorithm caused by incorrect sensor measurements can lead the robot in a wrong direction. The performance of the localization algorithm depends heavily on accuracy of the sensor system. The more precise the sensor measurements are carried out, the better position estimates are obtained. The graph in Figure A.11 compares the optimal distance (computed by the CSP planner) and the real covered trajectory as the function of laser range sensors. The average of ten simulation runs is taken.

As can be seen from the graph, with precise sensors, the average loss on optimum solution was about 5%. In this particular case, the localization component worked flawlessly and no plan recreation by the CSP planner was needed at all.

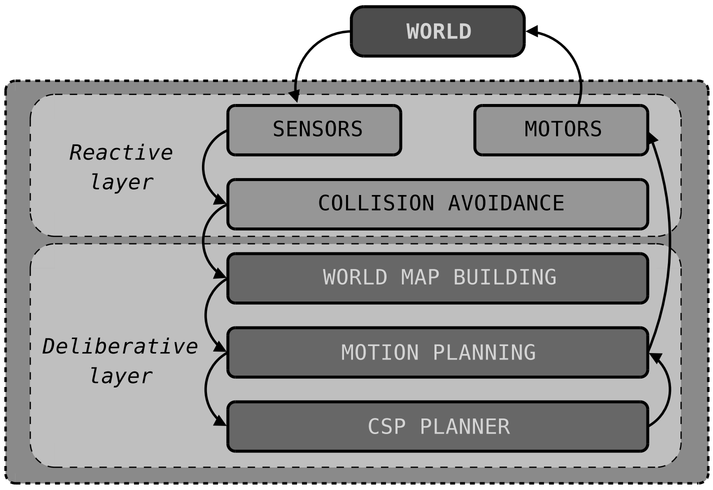


Figure A.10: Overall system architecture from bird's eye view.

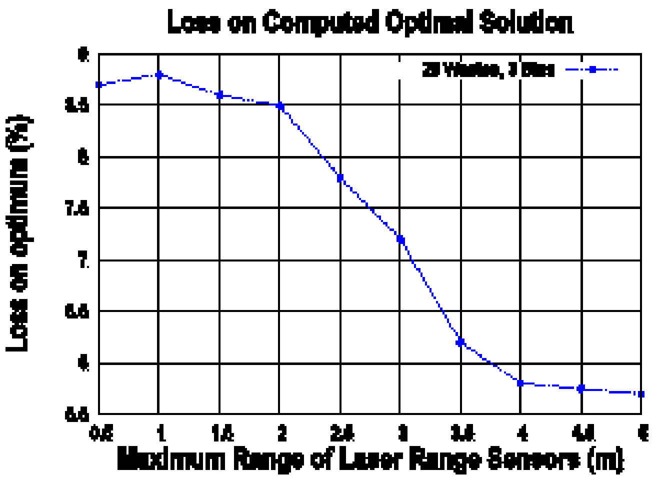


Figure A.11: Comparison of the optimal distance (computed by CSP planning algorithm) and real covered distance as a function of the sensor range.

A.8 Conclusions

We developed the robotic architecture incorporating both purely reactive execution and deliberative planning that works in complex and dynamic environment. The goal of the robot is to pick up all wastes in a given environment and put them to collectors while assuming a limited capacity of the robot. We used a constraint model based on network flows that is traditionally applied to this type of routing problems and we developed a completely new model based on finite automata. Using the constraint programming techniques allowed us to naturally define the underlying model for which the solver was able to find the first solution in hundreds of microseconds on problems of reasonable size. We further studied local search techniques that are traditionally used to improve the runtime performance of CP models for vehicle routing problems and we have found that our novel model based on finite automata performs better without local search. The preliminary experiments showed some interesting behaviour of the model in relation to the number of collectors that we are going to further investigate.

An important aspect of the presented work is the integration into a simulation environment describing real robots in realistic worlds. This integration showed that it is viable to use sophisticated planning methods together with reactive techniques.

In summary, there are three novel contributions. First, we reformulated the traditional network flow model to solve the waste collecting problem with limited capacity of the robot. Second, we proposed a novel constraint model based on finite automata (state transitions) and we experimentally showed that it outperforms the traditional approach, if the number of waste collecting places is small. Finally, we integrated the proposed models with a reactive planner to show that deliberative planning based on CP can be used in real robots and environments.

Bibliography

[1] Ajith Abraham, Rajkumar Buyya, and Baikunth Nath. Nature's Heuristics for Scheduling Jobs on Computational Grids. In *IEEE International Conference on Advanced Computing and Communications*, pages 45–52, 2000.

[2] STAR Collaboration: J. Adams. Experimental and theoretical challenges in the search for the quark gluon plasma: The STAR collaboration's critical assessment of the evidence from RHIC collisions. *Nuclear Physics A*, 757:102, 2005.

[3] Giorgio Ausiello, M. Protasi, A. Marchetti-Spaccamela, G. Ga mbosi, P. Crescenzi, and V. Kann. *Complexity and Approximation: Combinatorial Optimization Pro blems and Their Approximability Properties*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[4] Jonathan F. Bard, Liu Huang, Moshe Dror, and Patrick Jaillet. A branch and cut algorithm for the VRP with satellite facilitie s. *IIE Transactions*, 30(9):821–834, 1998.

[5] Roman Barták, Michal Zerola, and Stanislav Slusny. Towards Routing for Autonomous Robots - Using Constraint Programming in an Anytime Path Planner. In Joaquim Filipe and Ana L. N. Fred, editors, *ICAART*, pages 313–320. SciTePress, 2011.

[6] J. Christopher Beck, Andrew J. Davenport, Edward M. Sitarski, and Mark S. Fox. Texture-based heuristics for scheduling revisited. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 241–248. AAAI Press, 1997.

[7] Belaid Benhamou. Study of symmetry in constraint satisfaction problems. In *Proceedings of Principles and Practice of Constraint Programming*, pages 246–254, 1994.

[8] Christian Bessiere, Emmanuel Hebrard, Brahim Hnich, Zeynep Kiziltan, Claude-Guy Quimper, and Toby Walsh. Reformulating global constraints: the slide and regular constraints. In *Proceedings of the 7th International conference on Abstraction, reformulation, and approximation*, SARA'07, pages 80–92, Berlin, Heidelberg, 2007. Springer-Verlag.

[9] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy. Task scheduling strategies for workflow-based applications in grids. In *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, volume 2, pages 759–767, 2005.

[10] Tracy D. Braun, Howard Jay Siegel, Noah Beck, Ladislau L. Boloni, Muthucumaru Maheswaran, Albert I. Reuther, James P. Robertson, Mitchell D. Theys, Bin Yao, Debra Hensgen, and Richard F. Freund. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. *Journal of Parallel and Distributed Computing*, 61(6):810 – 837, 2001.

[11] Junwei Cao, Stephen A. Jarvis, Subhash Saini, and Graham R. Nudd. Gridflow: Workflow management for grid computing. In *Proceedings of the 3st International Symposium on Cluster Computing and the Grid*, CCGRID '03, pages 198–, Washington, DC, USA, 2003. IEEE Computer Society.

[12] Petr Chaloupka, Pavel Jakl, Jan Kapitán, Michal Zerola, Jérôme Lauret, and the Star collaboration. Setting up a STAR Tier 2 Site at Golias/Prague Farm. *Journal of Physics: Conference Series*, 219(7):072031, 2010.

[13] Ann Chervenak, Ian Foster, Carl Kesselman, Charles Salisbury, and Steven Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of Network and Computer Applications*, 23:187–200, 1999.

[14] Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, and Judith Stafford. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley, Boston, MA, 2003.

[15] Petrica C.Pop. New Integer Programming Formulations of the Generalized T ravelling Salesman Problem. *American Journal of Applied Sciences*, 11:932–937, 2007.

[16] Shaul Dar, Michael J. Franklin, Björn T. Jónsson, Divesh Srivastava, and Michael Tan. Semantic data caching and replacement. In *Proceedings of the 22th International Conference on Very Large Data Bases*, VLDB '96, pages 330–341, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[17] Rina Dechter. *Constraint Processing*. Morgan Kaufmann Publishers, San Francisco, CA, USA, May 2003.

[18] Peter J. Denning. The locality principle. *Communications of the ACM - Designing for the mobile device*, 48:19–24, July 2005.

[19] FDT. http://monalisa.cern.ch/FDT.

[20] Hanhua Feng, Vishal Misra, and Dan Rubenstein. PBS: a unified priority-based scheduler. In *Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS '07, pages 203–214, New York, NY, USA, 2007. ACM.

[21] Marshall L. Fisher. The Lagrangian Relaxation Method for Solving Integer Programming Problems. *Management Science*, 27(1):1–18, January 1981.

[22] B. G. Gibbard and T. G. Throwe. The RHIC computing facility. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 499(2-3):814 – 818, 2003. The Relativistic Heavy Ion Collider Project: RHIC and its Detectors.

[23] GLPK. http://www.gnu.org/software/glpk/.

[24] GLPK-java. http://glpk-java.sourceforge.net/.

[25] Solomon W. Golomb and Leonard D. Baumert. Backtrack programming. *J. ACM*, 12(4):516–524, 1965.

[26] R. Graham, E. Lawler, J. Lenstra, and A. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann. Discrete Math.*, 5:169–231, 1979.

[27] H. Hahn, E. Forsyth, H. Foelsche, M. Harrison, J. Kewisch, G. Parzen, S. Peggs, E. Raka, A. Ruggiero, A. Stevens, S. Tepikian, P. Thieberger, D. Trbojevic, J. Wei, E. Willen, S. Ozaki, and S. Y. Lee. The RHIC design overview. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers,*

Detectors and Associated Equipment, 499(2-3):245 – 263, 2003. The Relativistic Heavy Ion Collider Project: RHIC and its Detectors.

[28] A. Hanushevsky, A. Dorigo, and F. Furano. The Next Generation Root File Server. In *Proceedings of the Computing in High Energy and Nuclear Physics (CHEP) conference*, pages 680–683, 2005.

[29] Holger H. Hoos and Thomas Stutzle. *Stochastic Local Search : Foundations & Applications (The Morgan Kaufmann Series in Artificial Intelligence)*. Morgan Kaufmann, 1 edition, September 2004.

[30] Pavel Jakl. Efficient access to distributed data: 'a many' storage element paradigm. Master's thesis, Faculty of Nuclear Sciences and Physical Engineering, Czech Technical University in Prague, 2008.

[31] Theodore Johnson and Dennis Shasha. 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm. In *Proceedings of Very Large Data Bases*, pages 439–450, 1994.

[32] Carl Kesselman and Ian Foster. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, November 1998.

[33] Dalibor Klusacek, Ludek Matyska, and Hana Rudova. Local Search for Deadline Driven Grid Scheduling. In *In Third Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS 2007)*, pages 74–81, 2007.

[34] R.L. Kruse. *Data structures and program design*. Prentice-Hall software series. Prentice-Hall, 1987.

[35] Jérôme Lauret and David Yu. ERADAT and DataCarousel systems at BNL: A tool and UI for efficient access to data on tape with fair-share policies capabilities. In *Advanced Computing and Analysis Techniques in Physics Research*, 2010.

[36] T. Ludlam. RHIC and Quark Matter: A Proposed Heavy Ion Collider at Brookhaven National Laboratory. In K. Kajantie, editor, *Quark Matter '84*, volume 221 of *Lecture Notes in Physics, Berlin Springer Verlag*, pages 221–240, 1985.

[37] Jiri Matousek and Bernd Gärtner. *Understanding and Using Linear Programming (Universitext)*. Springer, 1 edition, November 2006.

[38] Nimrod Megiddo and Dharmendra S. Modha. ARC: A Self-Tuning, Low Overhead Replacement Cache. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, pages 115–130, Berkeley, CA, USA, 2003. USENIX Association.

[39] Young Jin Nam and Chanik Park. An adaptive high-low water mark destage algorithm for cached raid5. In *PRDC'02*, pages 177–184, 2002.

[40] Moni Naor and Udi Wieder. A simple fault tolerant distributed hash table. In M. Frans Kaashoek and Ion Stoica, editors, *IPTPS*, volume 2735 of *Lecture Notes in Computer Science*, pages 88–97. Springer, 2003.

[41] George L. Nemhauser and Laurence A. Wolsey. *Integer and combinatorial optimization*. Wiley-Interscience, New York, NY, USA, 1988.

[42] J. Packard, D. Katramatos, J. Lauret, K. Shroff, J. DeStephano, M. Ernst, J. Hover, T. Ichihara, D. Kim, S. McKee, M. L. Purschke, Y. Watanabe, J. Woo, I. Yoo, and D. Yu. High performance data transfer and monitoring for RHIC and USATLAS. *Journal of Physics: Conference Series*, 219(6):062062, 2010.

[43] Andrew J. Page and Thomas J. Naughton. Framework for Task Scheduling in Heterogeneous Distributed Computing Using Genetic Algorithms. *Artificial Intelligence Review*, 24:137–146, 2004.

[44] Claude Le Pape, Philippe Couronne, Didier Vergamini, and Vincent Gosselin. Time-versus-capacity compromises in project scheduling. In *Proceedings of the Thirteenth Workshop of the U.K. Planning Special Interest Group*, 1994.

[45] Gilles Pesant. A Regular Language Membership Constraint for Finite Sequences of Variables. In *Principles and Practice of Constraint Programming*, pages 482–495, 2004.

[46] Rashedur M. Rahman, Ken Barker, and Reda Alhajj. Study of Different Replica Placement and Maintenance Strategies in Data Grid. In *CCGRID*, pages 171–178. IEEE Computer Society, 2007.

[47] Kavitha Ranganathan and Ian Foster. Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications. In *11th IEEE Symposium on High-Performance Distributed Computing*, volume 0, pages 352–358, Los Alamitos, CA, USA, 2002. IEEE Computer Society.

[48] Krzysztof R.Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.

[49] A. Rasooli, M. Mirza-Aghatabar, and S. Khorsandi. Introduction of novel dispatching rules for grid scheduling algorithms. In *International Conference on Computer and Communication Engineering*, ICCCE 2008, pages 1072 –1078, may 2008.

[50] Jean-Charles Régin. Generalized arc consistency for global cardinality constraint. In *Proceedings of the thirteenth national conference on Artificial intelligence - Volume 1*, AAAI'96, pages 209–215. AAAI Press, 1996.

[51] Graham Ritchie and John Levine. A fast, effective local search for scheduling independent jobs in heterogeneous computing environments. In *Proceedings of the 22nd Workshop of the UK Planning and Scheduling Special Interest Group*, PLANSIG '03, pages 178–183, 2003.

[52] Louis-Martin Rousseau, Michel Gendreau, and Gilles Pesant. Using Constraint-Based Operators to Solve the Vehicle Routing Problem with Time Windows. *Journal of Heuristics*, 8:43–58, January 2002.

[53] N. Saito. Spin physics program at RHIC: the first polarized-proton collider. *Nuclear Physics B - Proceedings Supplements*, 105(1-3):47 – 51, 2002.

[54] Hitoshi Sato, Satoshi Matsuoka, Toshio Endo, and Naoya Maruyama. Access-Pattern and Bandwidth Aware File Replication Algorithm in a Grid Environment. In *GRID*, pages 250–257. IEEE, 2008.

[55] Srinath Shankar and David J. DeWitt. Data Driven Workflow Planning in Cluster Management Systems. In *HPDC '07: Proceedings of the 16th international symposium on High performance distributed computing*, pages 127–136, New York, NY, USA, 2007. ACM.

[56] Helmut Simonis. Constraint applications in networks. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 25, pages 875–903. Elsevier, 2006.

[57] Stanislav Slusny, Michal Zerola, and Roman Neruda. Real time robot path planning and cleaning. In De-Shuang Huang, Xiang Zhang, Carlos A. Reyes García, and Lei Zhang, editors, *ICIC (2)*, volume 6216 of *Lecture Notes in Computer Science*, pages 442–449. Springer, 2010.

[58] Danny Teaff, Dick Watson, and Bob Coyne. The Architecture of the High Performance Storage System (HPSS). In *Proceedings of the Goddard Conference on Mass Storage and Technologies*, pages 28–30, 1995.

[59] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed Computing in Practice: The Condor Experience. *Concurrency and Computation: Practice and Experience*, 17:2–4, 2005.

[60] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128(1-2):99–141, 2000.

[61] Vadim G. Timkovsky. Is a unit-time job shop not easier than identical parallel machines? *Discrete Applied Mathematics*, 85(2):149–162, 1998.

[62] Stephen H. Unger. Hazards, Critical Races, and Metastability. *IEEE Transactions on Computers*, 44:754–768, 1995.

[63] Petr Vilím, Roman Barták, and Ondřej Čepek. Extension of $O(n \log n)$ filtering algorithms for the unary resource constraint to optional activities. *Constraints*, 10(4):403–425, 2005.

[64] Fatos Xhafa and Ajith Abraham. Computational models and heuristic methods for grid scheduling problems. *Future Generation Computer Systems*, 26(4):608 – 621, 2010.

[65] Fatos Xhafa, Javier Carretero, Bernabe Dorronsoro, and Enrique Alba. A Tabu Search Algorithm for Scheduling Independent Jobs in Computational Grids. *Computing and Informatics*, pages 237–250, 2009.

[66] M.Q. Xu. Effective metacomputing using LSF Multicluster. In *First IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 100 –105, 2001.

[67] Asim Yarkhan and Jack J. Dongarra. Experiments with Scheduling Using Simulated Annealing in a Grid Environment. In *GRID 2002*, pages 232–242, 2002.

[68] Jia Yu and Rajkumar Buyya. A Taxonomy of Workflow Management Systems for Grid Computing. *Journal of Grid Computing*, 3(3):171–200, September 2005.

[69] Michal Zerola, Roman Barták, Jérôme Lauret, and Michal Šumbera. Using constraint programing to resolve the multi-source / multi-site data movement paradigm on the grid. In *Advanced Computing and Analysis Techniques in Physics Research. PoS(ACAT08)039*, 2008.

[70] Michal Zerola, Roman Barták, Jérôme Lauret, and Michal Šumbera. Efficient Multi-site Data Movement in Distributed Environment. In *Proceedings of the* 10^{th} *IEEE/ACM International Conference on Grid Computing (GRID)*, pages 171–172. IEEE, 2009.

[71] Michal Zerola, Roman Barták, Jérôme Lauret, and Michal Šumbera. Planning Heuristics for Efficient Data Movement on the Grid. In *Proceedings of the* 4^{th} *Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA)*, pages 768–771, 2009.

[72] Michal Zerola, Roman Barták, Jérôme Lauret, and Michal Šumbera. Using Constraint Programming to Plan Efficient Data Movement on the Grid. In *Proceedings of the* 21^{st} *IEEE International Conference on Tools with Artificial Intelligence*, pages 729–733. IEEE Computer Society, 2009.

[73] Michal Zerola, Roman Barták, Jérôme Lauret, and Michal Šumbera. Building Efficient Data Planner for Peta-scale Science. In *Advanced Computing and Analysis Techniques in Physics Research. PoS(ACAT10)025*, 2010.

[74] Yuanyuan Zhou, James Philbin, and Kai Li. The Multi-Queue Replacement Algorithm for Second Level Buffer Caches. In *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, pages 91–104, Berkeley, CA, USA, 2001. USENIX Association.

[75] Albert Y. Zomaya and Yee-Hwei Teh. Observations on Using Genetic Algorithms for Dynamic Load-Balancing. *IEEE Trans. Parallel Distrib. Syst.*, 12:899–911, September 2001.