Title

Heuristics for Scheduling Parameter Sweep Applications in Grid Environments

Authors

Casanova, Henri
Legrand, Arnaud
Zagorodnov, Dmitrii
et al.

Publication Date

1999-10-14

Peer reviewed

Heuristics for Scheduling Parameter Sweep Applications in Grid Environments [1]

Henri Casanova [2] Arnaud Legrand [3]

Dmitrii Zagorodnov [2] Francine Berman [2]

Abstract

The computational Grid provides a promising platform for the efficient execution of *parameter sweep applications* over very large parameter spaces. Scheduling such applications is challenging because target resources are heterogeneous, because their load and availability varies dynamically, and because tasks may share common data files. In this paper, we propose a scheduling algorithm for parameter sweep applications on the Grid. We consider standard heuristics for task/host assignment (*Max-min*, *Min-min*, *Sufferage*), and we propose an extension of Sufferage called *XSufferage*. Using simulation, we demonstrate 3 results: 1) that XSufferage can take advantage of file sharing to achieve better performance than the other heuristics under a wide variety of load conditions, 2) that it is possible to characterize the environments under which different heuristics perform best, and 3) that it is possible to characterize the performance of different heuristics under the (realistic) assumption of varying accuracy of performance estimations.

[1]This research was supported in part by NSF Grant ASC-9701333, NASA/NPACI Contract AD-435-5790, and DARPA/ITO under contract #N66001-97-C-8531.

[2]Department of Computer Science and Engineering, University of California San Diego, USA

[3]École Normale Supérieure de Lyon, France

1 Introduction

Fast networks make it possible to aggregate CPU, network and storage resources into computational *Grids* [1]. Such environments can be used effectively to support very large-scale runs of distributed applications. An ideal class of applications for the Grid is the class of *parameter sweep applications*, applications structured as a set of multiple "experiments", each of which is executed with a distinct set of parameters.

Executing a parameter sweep on the Grid involves the assignment of tasks to resources. Although the experiments (or *tasks*) of a parameter sweep application are independent, a number of issues make scheduling such applications challenging. First, resources on the Grid are typically *shared* so that the contention created by multiple applications creates dynamically fluctuating delays and qualities of service. In addition, Grid resources are *heterogeneous* and may not perform similarly for the same application. Moreover, although parameter sweep tasks are independent, they may share common input files which reside at remote locations, hence the performance-efficient assignment and scheduling of the application must include consideration of the impact of data transfer times.

In [2], 3 heuristics (*Max-min, Min-min* and *Sufferage*) were proposed for the scheduling of independent tasks in single-user, homogeneous environments. **In this paper, we modify these heuristics to schedule parameter sweep applications in shared, heterogeneous environments and we propose an extended version of Sufferage, XSufferage**. We characterize the environments in which each heuristic is performance-efficient and discuss how a scheduler might adapt its policy based on system characteristics, prediction accuracy, size of shared files, and other environmental parameters to promote application performance. We use a classic performance metric: the application *makespan* [3], i.e. the time until until all application output is available to the user.

Our ultimate goal is to include our scheduling algorithm in adaptive software designed to dynamically schedule Grid parameter sweep applications. Such software is being developed as part of the AppLeS project [4, 5, 6] and will be the topic of a future paper. The work presented here is a first step in the development of a parameter sweep Grid application scheduler.

This paper is organized as follows. In Section 2, we present our models for both the application and the underlying Grid environment. In Section 3, we present our scheduling algorithm. Section 4 discusses the different task/host assignment heuristics. Section 5 provides simulation results, Section 6 references related research works, and Section 7 concludes the paper.

2 A Scheduling Model for Parameter Sweeps on the Grid

We define a **parameter sweep** application as a set of n independent sequential tasks (or *experiments*) $\{T_i\}_{i=1,..,n}$. By *independent* we mean that there are no inter-task communications or data dependencies (i.e. task precedences). We assume that the input to each task is a set of files and that a file might be input to more than one task. In our model, without loss of generality, each task produces exactly one output file. Figure 1 shows an example with input file sharing among tasks. We assume that the size of each input and output file is known a-priori.

We view the Grid computing environment available to the user as a set of k clusters of computing resources $\{C_j\}_{j=1,..,k}$ that are accessible via k network links. Each cluster contains a

certain number of *hosts* where a host can be any computing platform, from a single-processor workstation to an MPP system, and is available in interactive or batch mode. We will call hosts and network links *resources*. For each computation and file transfer, we assume an estimate of execution time performance is available. On interactive hosts, the estimate is the application *execution time* whereas on batch resources, it is is the *turnaround time* (defined as the queue waiting time plus execution time). Such estimates can be provided by the user, computed from analytical models or historical information, or provided by facilities such as the Network Weather Service (NWS) [7], ENV [8], Remos [9], and/or Grid services such as those found in Globus [10]. Note that the accuracy and variance of the performance estimates can provide an additional information. In this work we initiate a study of the impact of estimate accuracy on different scheduling strategies.

We assume that a storage facility (e.g. NFS) is available at each cluster so that files can be shared among the processes running on different hosts in the cluster[4]. Figure 2 shows an example with three clusters. In this work we assume that all input files are initially stored on the user's host, that all output files must be returned to this location, and that there are no inter-cluster file exchanges. We assume for now that once assigned, tasks do not migrate between resources. This scenario fits the current usage of several real-life, parameter sweep applications (e.g. MCell [11, 12], INS2D [13, 14] and others), and we leave alternate usage scenarios for future work. The models assumptions will be further discussed in the full paper and we believe that our simplifying assumptions make it possible to obtain initial meaningful results about a realistic environment while keeping the simulation tractable.

3 A Parameter Sweep Scheduling Algorithm

Our scheduling algorithm, `schedule()`, takes into account performance estimates to generate a *plan* for the scheduling of tasks that have not yet been assigned to hosts. To account for the Grid's dynamic nature, `schedule()` can be called repeatedly so that the schedule can be modified and refined. This is depicted on Figure 3. We denote the points in time at which `schedule()` is called *scheduling events*, according to the terminology in [2]. We assume that at each scheduling event our scheduler has access to: (i) the current topology of the Grid (number of clusters, number of hosts in those clusters, network links), (ii) the number and location of copies of all input files, and (iii) the list of computations and file transfers currently underway.

Figure 4 shows the general skeleton for `schedule()` whose steps can be described as follows: **Step (1)** determines the time of the next scheduling event. This can take into account environment behavior to increase or decrease the scheduling event frequency. **Step (2)** creates a Gantt chart [15] that will be used to keep track of task/host assignments. The Gantt chart contains as many columns as resources. Figure 5 shows an example for an environment containing two clusters with respectively two and three hosts. **Step (3)** inserts slots corresponding to tasks that are currently running into the chart. Two examples are shown on Figure 5 as black-filled rectangles slots at the beginning of the chart (one file transfer and one computation).

Step (5) is the core of the algorithm in which it is determined which task should be performed on which host. This step is detailed in Section 4. Examples of slot assignments are depicted

[4]For the moment we do not model contention arising from access to shared files within a cluster.

on Figure 5 in gray. In this example, input file transfers are scheduled on the network link to cluster C_2, the computation is then scheduled on a host within that cluster, and the output file is scheduled to be returned to the user's host. **Step (6)** converts the Gantt chart into a sequence of instructions. These instructions can then be followed to interact with Grid software services (for job submission and monitoring, data motion, etc.).

4 Task/Host Assignment Heuristics

Almost every step of our scheduling algorithm can use a variety of techniques. The ultimate goal of our research is to produce an adaptive algorithm that takes into account Grid and application information to choose appropriates techniques at each step. This paper is focused on step (5), the core of the algorithm: task-host selection heuristics. We must identify heuristics that are applicable in Grid environments. Moreover task/host assignment heuristics must be efficient to compute, since the scheduler must not take longer to determine a schedule than the performance benefits of using any individual schedule in practice.

Three simple heuristics for scheduling independent tasks for a uniform single-user environment are proposed in [2]: *Min-min*, *Max-min*, and *Sufferage*. They are based on work in [16]. *Min-min* computes each task's Minimum Completion Time (MCT) over the available hosts and the task with the minimum MCT is assigned to its best host. The motivation behind Min-min is that assigning tasks to hosts that complete them fastest will lead to an overall reduced makespan. Similarly, *Max-min* assigns the task with maximum MCT to its best host. The expectation is to overlap long-running tasks with short-running ones. Finally, the rationale behind *Sufferage* is that a host should be assigned to the task that would suffer the most if not assigned to that host (i.e. by an alternative assignment). The task with the highest *sufferage value* (difference between its best and second-best completion time) takes precedence. More details on these heuristics can be found in [2].

We modified all 3 heuristics so that they (i) include input and output data transfer times when computing MCT and (ii) take into account the fact that some files may already be present on remote storage devices. In addition, we implemented an extended version of the Sufferage heuristic, *XSufferage*. In XSufferage, the sufferage value is computed not with MCTs, but with *cluster-level MCTs*, i.e. by computing the minimum MCTs over all hosts in each cluster. The task with the highest cluster-level sufferage values is then assigned to the host that achieves the lowest MCT, in the cluster that achieves the lowest cluster-level MCT. The full paper will give detailed algorithmic descriptions of all the heuristics we are considering.

It is reasonable to expect that a sufferage strategy should lead to good performance in the shared and heterogeneous setting, since the sufferage value provides a convenient and simple way to reflect presence of input files on remote clusters. The main idea is that a task would "suffer" the least if it is assigned to a host on a cluster that already has access to some of its large input files. This is somewhat reminiscent of the idea of *task/host affinities* introduced in [2], where some hosts are better for some tasks but not for others.

5 Simulation Results

In order to evaluate the efficiency of these heuristics we developed a simulator. The simulator allows us to compare heuristics under the same load conditions representing a wide variety of system states. In addition, we verified the accuracy of our simulated results by comparing experimental runs in shared, production environments with similarly loaded simulation application execution times. Our simulator takes as input `schedule()`, a task/host assignment heuristic, a description of the application tasks and input/output files, and a description of the Grid topology with performance characteristics of Grid resources (constant values, samples from random variables, traces from the NWS [7])[5]. The output of the simulator is a makespan value based on the set of input parameters. More details on the simulator can be found in [17].

Figures 6(a) and (b) show simulation results for an application that consists of 400 tasks. Each task takes in input a 10K un-shared file and one of eight identical shared files, each shared by 50 tasks. All tasks are identical in terms of computational cost and produce a 10K output file. The graphs plot makespan vs. shared file size for all four heuristics and for a *self-scheduled workqueue* strategy [18]. This application setting is comparable to what some of our target parameter sweep applications (e.g. MCell [11, 12]) require. The simulated Grid consists of 4 clusters containing respectively 4, 5, 6 and 8 hosts. CPU loads, network latencies and bandwidths are modeled by various traces from actual NWS measurements making this Grid heterogeneous. (The full paper will give all the details about simulation settings). For large shared file sizes on the graphs, the average ratio between file transfer time and computation time for one task is about 40. This means that on average it is 40 times more expensive to send a shared file across the network than to run one task. For the smallest shared file sizes in our study, that ratio is about 0.05. For these experiments, the interval between scheduling events was always 300 seconds, and we assumed 100% accurate performance estimations.

Figure 6(a) shows the makespan for "small" shared files. One can observe that the self-scheduled workqueue leads to better makespan than other heuristics when files are so small that the effect of file sharing becomes negligible. However, the workqueue quickly becomes inefficient when shared file size increases. One can also observe on that graph that Max-min performs poorly. According to results in [2], this must be due to the fact that our tasks all have identical computational costs. The remaining three heuristics seem to perform similarly for small shared file sizes. Figure 6(b) shows the results for large file sizes. In that range XSufferage performs about 30% better than Min-Min and we believe that this is because it can better capture information about file sharing with respect to storage resources. The Sufferage heuristic performs very poorly (even worse than Max-min for large files) due to the inability of host-level sufferage values to exploit input file locations. In order to make these results more general and to show how different heuristics are best for different types of Grid systems and applications, the full paper will contain data from similar experiments in a wider range of settings.

A new avenue of research that we have mentioned earlier is the study of *quality of information* [19] on scheduling, that is the impact of the performance estimation accuracy on different scheduling strategies. We expect different task/host selection heuristics to react differently to degrading levels of accuracy and that strategies that do not depend on performance estimation and forecast (e.g. self-scheduled ones [20]) may be more appropriate in situations when quality of

[5]The simulator also allows for transient resources.

information is low. Figure 7 shows simulation results for the same environment as above and for a shared file size of 36 MBytes. In this experiment we introduced noise to the perfectly accurate estimates of the previous experiment by adding a random percentage error to the forecasts (positive, uniformly distributed). As the mean noise percentage increases, this simulates increasingly pessimistic forecasts (over-estimations) of the task computation times and file transfer times. On the graph we plot the makespan vs. different values of that maximum percentage error and for all four heuristics and the self-scheduled workqueue. One can see that XSufferage seems to the most tolerant to bad quality of information. The Max-min and Sufferage heuristics quickly become less efficient than the workqueue whose makespan does not depend on quality of information. This first result is encouraging and motivates future work in the study of quality of information with respect to scheduling algorithms.

6 Related Work

A large number of research works attack the question of mapping set of tasks onto set of processors in a view to minimizing overall execution time. Many of these works focuses on sets of independent tasks [16, 18, 20, 2]. Scheduling heuristics found in [2] were adapted to our framework as discussed in Section 4. Our contribution is that we pay special attention to shared application data, heterogeneous target resources, and accuracy of information.

A number of recent projects [21, 22] address the question of simulating heterogeneous distributed environment for the purpose of evaluating scheduling strategies. Few actual implementations are underway but Bricks [21] should become available in the near future. However, Bricks targets "global computing systems" [23, 24, 25, 26, 10] rather than application schedulers. It assumes constant task and data arrival rates to servers and uses queuing theory, in an attempt to model many users who asynchronously interact with a global computing system. By contrast, our simulator is purely event-driven which is more appropriate in our framework where the scheduler knows all tasks a-priori and is in charge of only one application.

7 Conclusion and Future Work

In this paper we have proposed a scheduling algorithm for parameter sweep applications in Grid environments. After precisely defining our application and Grid model, we adapted three standard heuristics for task/host assignment (*Max-min*, *Min-min*, *Sufferage*) and proposed an extension of the *Sufferage* heuristic, *XSufferage*. Initial simulation results demonstrated that: (i) XSufferage can lead to efficient schedules in the case of input file sharing among tasks; (ii) XSufferage seems to be more tolerant of inaccurate resource performance forecasts and estimations.

A number of avenues are open for future research. Those include techniques for determining the optimal scheduling event frequency, more investigation of the impact of quality of information, exploration of alternate scheduling heuristics (e.g. randomized heuristics [27, 28]),variations in the initial location of input data and post-processing of output data, as well as other performance metrics. The work described herein forms the basis for a practical software project that aims at providing a programming environment for parameter sweep applications (including a scheduler) in shared, heterogeneous Grid environments.

References

[1] Ian Foster and Carl Kesselman, editors. *The Grid, Blueprint for a New computing Infrastructure*. Morgan Kaufmann Publishers, Inc., San Francisco, USA, 1998.

[2] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. Freund. Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems. In *8th Heterogeneous Computing Workshop (HCW'99)*, Apr. 1999. to appear.

[3] Pinedo M. *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall, Englewood Cliffs, NJ, 1995.

[4] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application-Level Scheduling on Distributed Heterogeneous Networks. In *Proc. of Supercomputing'96, Pittsburgh*, 1996.

[5] F. Berman and R. Wolski. The AppLeS Project: A Status Report. In *Proc. of the 8th NEC Research Symposium, Berlin, Germany*, May 1997.

[6] F. Berman, R. Wolski, and G. Shao. Performance Effects of Scheduling Strategies for Master/Slave Distributed Applications. Technical Report TR-CS98-598, U. C., San Diego, 1998.

[7] R. Wolski. Dynamically Forecasting Network Performance Using the Network Weather Service. In *6th High-Performance Distributed Computing Conference*, pages 316–325, August 1997.

[8] G. Shao, F. Breman, and R. Wolski. Using Effective Network Views to Promote Distributed Application Performance. In *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications*, 1999.

[9] B. Lowekamp, N. Miller, D. Sutherland, T. Gross, P. Steenkiste, and J. Subhlok. A Resource Query Interface for Network-Aware Applications. In *Proceedings of the 7th IEEE Smposium on High-Performance Distributed Computing*, July 1998.

[10] I. Foster and K Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[11] J.R. Stiles, T.M. Bartol, E.E. Salpeter, , and M.M. Salpeter. Monte Carlo simulation of neuromuscular transmitter release using MCell, a general simulator of cellular physiological processes. *Computational Neuroscience*, pages 279–284, 1998.

[12] J.R. Stiles, D. Van Helden, T.M. Bartol, E.E. Salpeter, , and M.M. Salpeter. Miniature endplate current rise times <100 microseconds from improved dual recordings can be modeled with passive acetylcholine diffusion form a synaptic vesicle. In *Proc. Natl. Acad. Sci. U.S.A.*, volume 93, pages 5745–5752, 1996.

[13] S. Rogers and D. Ywak. Steady and Unsteady Solutions of the Incompressible Navier-Stokes Equations. *AIAA Journal*, 29(4):603–610, Apr. 1991.

[14] S. Rogers. A Comparison of Implicit Schemes for the Incompressible Navier-Stokes Equations with Artificial Compressibility. *AIAA Journal*, 33(10), Oct. 1995.

[15] W. Clark. *The Gantt chart*. Pitman and Sons, London, 3rd edition, 1952.

[16] O. H. Ibarra and C. E. Kim. Heuristic algorithms for scheduling independent tasks on nonindentical processors. *Journal of the ACM*, 24(2):280–289, Apr. 1977.

[17] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Using Simulation to Evaluate Scheduling Heuristics for a Class of Applications in Grid Environments. Technical report, École Normale Supérieure de Lyon, LIP, 1999. to appear.

[18] T. Hagerup. Allocating Independent Tasks to Parallel Processors: An Experimental Study. *Journal of Parallel and Distributed Computing*, 47:185–197, 1997.

[19] `http://www-cse.ucsd.edu/groups/hpcl/apples/perf.html`.

[20] Susan Flynn Hummel, Jeanette Schmidt, R. N. Uma, and Joel Wein. Load-sharing in heterogeneous systems via weighted factoring. In *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 318–328, June 1996.

[21] A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, and U. Nagashima. Overview of a performance evaluation system for global computing scheduling algorithms. In *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC8)*, Aug 1999. to appear.

[22] `http://www.hsdal.ufl.edu/Projects/Osculant/`.

[23] H. Casanova and J. Dongarra. NetSolve: A Network Server for Solving Computational Science Problems. *The International Journal of Supercomputer Applications and High Performance Computing*, 1997.

[24] S. Sekiguchi, M. Sato, H. Nakada, S. Matsuoka, and U. Nagashima. Ninf : Network based Information Library for Globally High Performance Computing. In *Proc. of Parallel Object-Oriented Methods and Applications (POOMA), Santa Fe*, pages 39–48, February 1996.

[25] P. Arbenz, W. Gander, and M. Oettli. The Remote Computational System. *Parallel Computing*, 23(10):1421–1428, 1997.

[26] A. Grimshaw, W. Wulf, J. French, A. Weaver, and P. Jr. Reynolds. A Synopsis of the Legion Project. Technical Report CS-94-20, Department of Computer Science, University of Virginia, 1994.

[27] M. Mitzenmacher. How useful is olf information. In *Proceedings of the 16th ACM Symposium on Principles of Distributed Computing*, pages 83–91, 1997.

[28] M. Mitzenmacher. On the Analysis of Randomized Load Balancing Schemes. *Theory of Computer Systems*, 32:361–386, 1999.

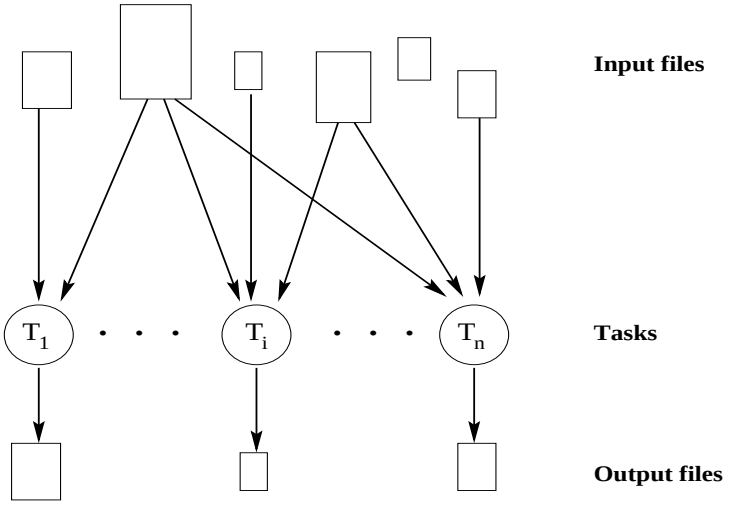

Figure 1: Application Model

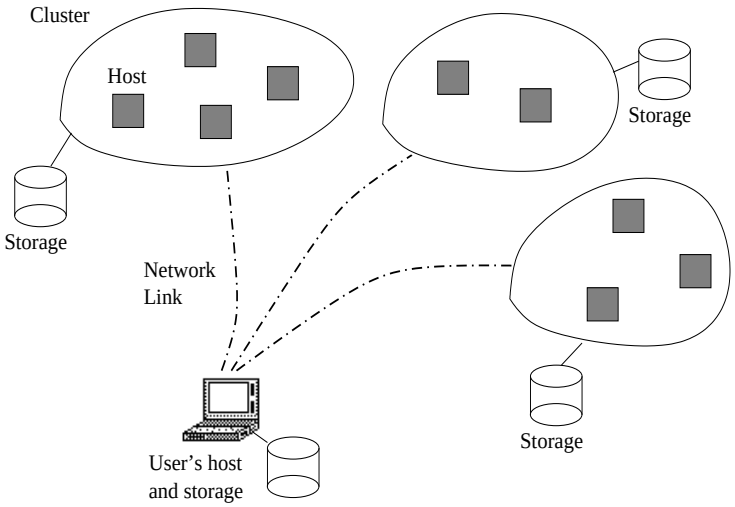

Figure 2: Environment Models

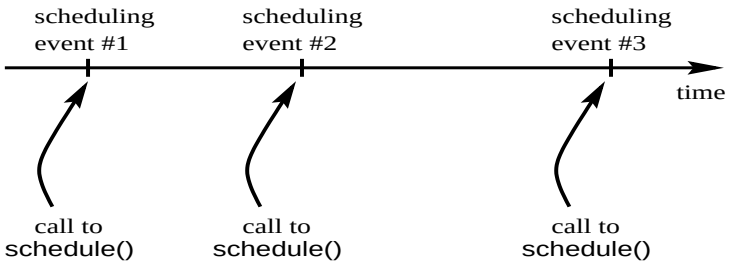

Figure 3: Scheduling scheme

```
schedule() {
    (1) compute the next scheduling event
    (2) create a Gantt Chart, G
    (3) foreach computation and file transfer currently underway
            compute an estimate of its completion time
            fill in the corresponding slots in G
    (5) until each host has been assigned enough work
        use heuristic to assign tasks to hosts, by filling slots in G
    (6) convert G into plan
}
```

Figure 4: Scheduling algorithm

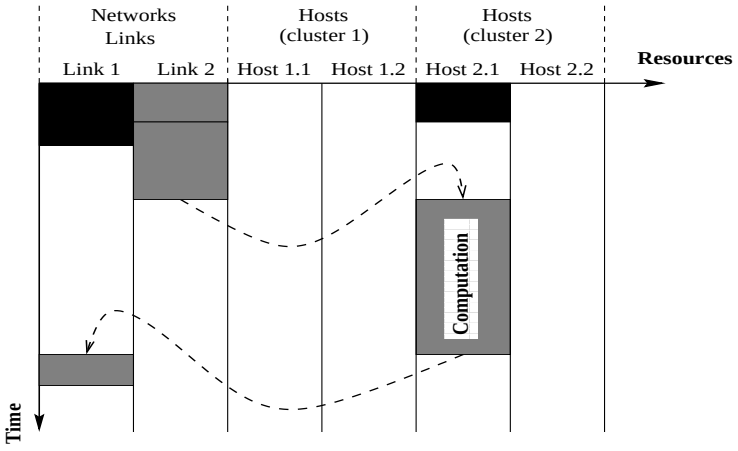

Figure 5: Sample Gantt chart

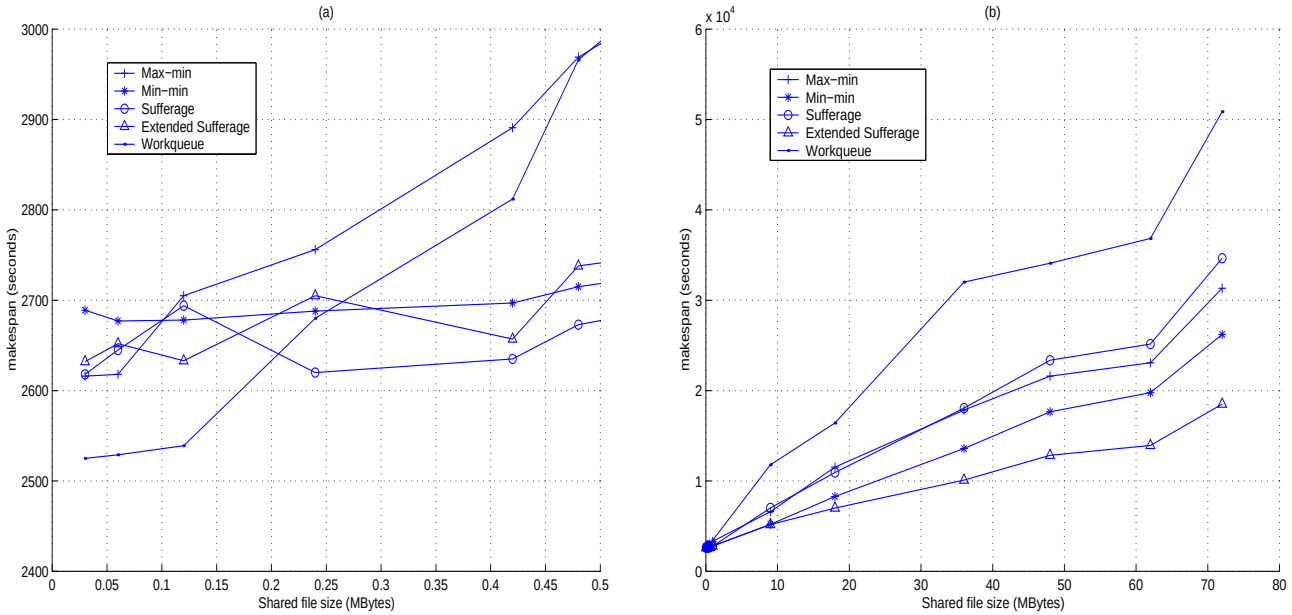

Figure 6: Makespan vs. shared file size for different heuristics

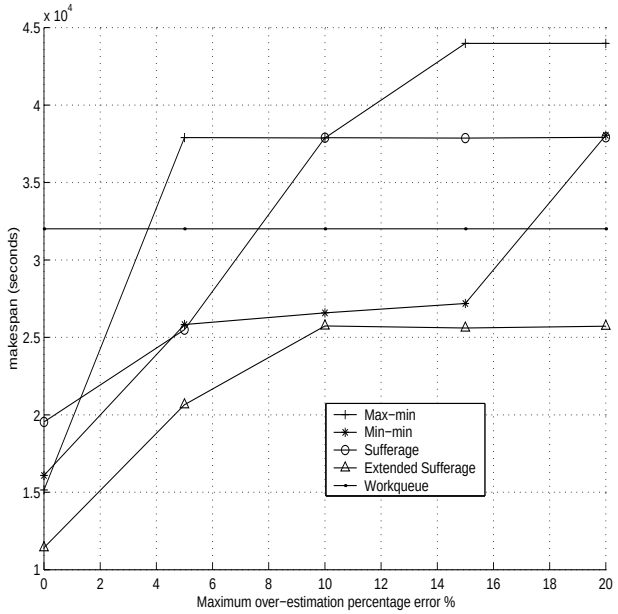

Figure 7: Makespan vs. quality of information