

Planning for Grid Applications with Explicit Reservations

Tatiana Kichkaylo

New York University
New York, NY 10012, USA
kichkay@cs.nyu.edu

Abstract

Computation and data intensive grid applications are often composed of independent components distributed over a wide-area network. Such components usually interact by consuming and producing files, which are transferred between network hosts using protocols such as GridFTP. An application configuration is defined by the choice of components, data files, hosts, and data transfer sessions, and depends on the current user goal, availability of data, and the state of the network. In a dynamic, heterogeneous environment such as modern grids it is desirable to construct application configurations at run time. In this paper we consider the problem of construction of application configurations when the resource availability in the network changes over time. We present a model for this problem and outline a planning algorithm that reasons about explicit reservations.

1 Introduction

Scientific applications that run on modern grids often process large amounts of data and involve long computations. Such applications usually consist of a set of independent components, each of which consumes and produces data files and runs on some network host (Figure 1). The files can be transferred between the hosts using protocols such as GridFTP.

A particular configuration of an application is defined by the choice of components, data files, hosts, and data transfer sessions. The best choice of application configuration depends on the current state of the grid. Some files required by the application might already be present in the grid. Hosts, which can run the required components, might experience different load levels that affect the completion time of the computation. The grid is a dynamic environment; therefore, it is desirable to construct application configuration at run time. We refer to the problem of run-time construction of application configurations given the state of the network as the application configuration problem (ACP). We use the term *plans* to denote application configurations, and refer to the algorithm that constructs them as a *planner*.

Grid applications often require specific equipment, such as supercomputers, and run for a long time. The

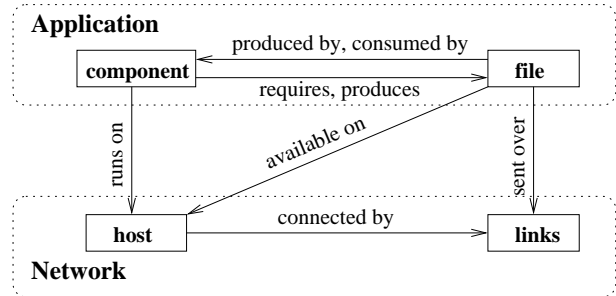


Figure 1: The entities of the problem.

combination of these two factors means that availability of the equipment changes over time. Usually, allocation of the high-demand equipment is done by special scheduling services. An automated planner can obtain from such services information about the availability of the equipment and book time slots to run application components. This allows the planner to produce better quality plans and reason about the completion time and the cost of the plan.

Traditionally, planners assume that all information is available to the planner from the very beginning. In the grid world, it is crucial to interact with services to get this information.

In this paper we present a model for the ACP with explicit reservations of host computation time. Our model also specifies the set of services from which a planner can obtain necessary information, and the functionality these services should provide. In Section 4 we outline a planning algorithm for solving the ACP with explicit reservations as a proof of viability of our model. We also discuss several modifications that would improve expressiveness of the model and performance of the algorithm.

2 Related Work

The problem of run-time construction of application configurations has been considered in the settings of dynamic component-based systems (Fu & Karamcheti 2003; Kichkaylo, Ivan, & Karamcheti 2003), web services (Wu *et al.* 2003), and grid computing (Blythe *et*

al. 2003). However, to the best of our knowledge, none of the existing planners supports explicit reservations of time slots.

Several planners (Golden, Etzioni, & Weld 1996; Nau *et al.* 2001) can invoke external services to obtain information. In the grid environment, planning operators and the information about the state of the world are not provided to the planner from the very beginning. Instead, they are generated at run time using data obtained from external services. Our work contributes by specifying a set of services sufficient to plan for grid applications.

Explicit reservations can be represented using timed initial literals introduced in PDDL 2.2. In principle, if all information is available to the planner from the very beginning, the ACP can be solved by a domain-independent planner such as SAPA (Do & Kambhampati 2003). However, since the domain and problem specifications need to be constructed on-the-fly, a domain-specific planner, such as the one described in this paper, is necessary.

Our work focuses on the search for the optimal application configuration, and views matchmaking (finding hosts capable of running components) and in-host scheduling (allocating time slots) problems as black boxes. We hope to interface our planner with existing matchmaking services such as (Tangmunarunkit, Decker, & Kesselman 2003) and (Paolucci *et al.* 2002).

3 The Model

The goal of the ACP is to construct an (optimal) application configuration that satisfies a set of constraints and achieves a user goal. The input to the ACP consists of an application description, a network description, and a goal. For simplicity, let us assume that the goal is to have a particular file on a given node. There might be additional restrictions on the budget (total or per-resource) and deadlines. Initially, the planner receives only the goal. The planner then obtains other information by calling external services. The rest of this section provides more details about the specification of the application components, network, external services, and plans.

3.1 Application description

An application consists of a set of components that require input files and produce output files.

The files used in a component description are logical. A logical file has a particular content and size. A logical file is uniquely identified by its logical file name (LFN). For each logical file there might be several physical copies (replicas) present in the network.

A component can run on a network host. A component description contains a specification of component's requirements such as the version of the OS running on the host, the number of processors, required disk space, etc. Following (Thain, Tannenbaum, & Livny 2002), we refer to such specifications as *ClassAds*. In our model, a *ClassAd* is considered a black box.

In addition, a component has a reference to functions for computing the cost and duration of computation given a network host on which the component runs. Invoked without a parameter, this function returns lower bounds on the cost and time.¹

3.2 Network description

The network consists of a set of hosts connected by network links.

Each host is described by a specification (*ClassAd*) of static information about the host such as hardware and the OS. In addition, each host runs a scheduling service that provides information about dynamic resources as described below.

While it is reasonable to assume that for each host there is some authority that has complete control over it, we have chosen not to make a similar assumption about the links of a wide-area network. For this reason, we do not model network links separately, but instead rely on a service that performs reservations at the granularity of paths and provides quality of service guarantees (see below).

3.3 Representation of plans

A plan is a DAG consisting of three types of nodes: file, component, and transfer. In a valid plan, each node is completely instantiated and has a *finite cost* and *completion time*. The DAG has one root file node corresponding to the user goal. The completion time of the plan is the completion time of its root node. The cost of the plan is the sum of costs of all nodes in the plan. A plan metric to be optimized is a combination of the cost and the completion time specified by the user.

File nodes of the plan describe availability of a data file (specified by its LFN) on a host. The completion time of a file node is the earliest time at which the physical file is available on the host.²

Component nodes correspond to execution of components on hosts. Each component node has a reference to the component, a reference to the host, and a time slot (reservation) during which the component runs on the host. Physical copies of all files required by the component need to be present on the node before the execution starts. The files produced by the component become available on the host upon completion of the execution.

File transfer nodes represent the operation of sending a physical file over the network. Thus, file transfer nodes connect file nodes. We do not require that a strict time slot be scheduled for each transfer. Instead, a file transfer node has a ticket to a network reservation service that guarantees that a given number of bytes is sent from one network host to another within a specified time window.

¹In the simplest case, the lower bounds are zero.

²Currently, we assume that files do not get removed from nodes. However, the model can be extended to reason about temporary storage of data.

3.4 External services

The planner can access information about the application, the network, and the goal using the following services.

Goal Supplier is the user agent that submits the goal as a pair (LFN, host). It can also supply deadline and budget restrictions.

Replica Location Service (RLS). RLS (Chervanek *et al.* 2002) is the service that for a given logical file name returns a (possibly empty) set of network hosts on which a physical copy of that file is present. For each such copy, RLS can also return the earliest availability time and the cost of access.

MatchMaker. The MatchMaker (Thain, Tannenbaum, & Livny 2002) keeps a list of ClassAds of hosts, and returns a list of matching hosts given a component's ClassAd. Note that the fact that a host was matched with a component does not necessarily mean that the component can be scheduled on that host.

Host Scheduler (HS) is a service that runs on each host. For a given component, HS returns a set of time slots during which the component can run. HS invokes the functions given in the component specification to compute the component's cost and completion time.

Application Registry (AR) is a catalog of components that can be used for application construction. For a given LFN, AR returns a set of components (component descriptions) that can produce this logical file.

Network Service (NS) computes the cost of data transfer and (possibly) the required time given the size of the file, the source and destination hosts, and the time window. We assume that NS minimizes the cost of the transfer for a fixed window, and the transfer time if the upper bound of the time window is not specified. This allows NS to optimize the overall network behavior by trading network routes and protocols for non-urgent transfers.

3.5 Example

As mentioned before, the planner obtains all information by querying external services. We present the following example in terms of decisions made by the services in response to queries issued by the planner.

Application. There are three component types (Figure 2). The *First* component requires file *F1* and produces files *F3* and *F4*. The *Second* component requires *F4* and produces *F5*. The *Third* component requires *F2*, *F3*, and *F5* and produces *F6*.

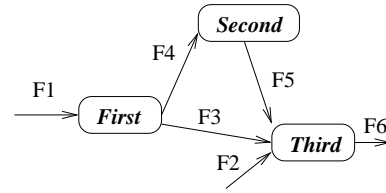


Figure 2: Example application.

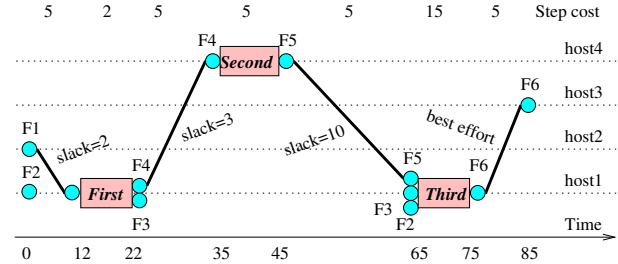


Figure 3: Example plan with cost 42 and completion time 85. Horizontal lines represent hosts. Circles show existence of a file on a node (the first time the file appears and the moment when it is required). Rectangles correspond to execution of a component. Solid lines are data transfers. In this example each transfer takes 10 time units, and the slack is shown near each line.

Network. The network consists of four hosts. File *F1* is initially available on *host2*, and *F2* on *host1*. The files are available there from the very beginning, and the access cost is zero.

Component *First* can run on *host1* starting at time points 2 and 12. The cost of running *First* on *host1* is 2, and its execution takes 10 time units in both cases. Component *Second* can run on *host4* starting at 10, 25, or 35. The execution cost is 5, and the required time is 10. Component *Third* can run on *host1* starting at 5, 45, or 65. The cost of execution of *Third* on *host1* is 15, and the required time is 10.

For any data transfer, the cost is 5 and the duration is 10. The Network Service is always willing to schedule a data transfer as long as the given time window is sufficiently big.

Goal. The goal is to have file *F6* available on *host3* with minimum cost. Among plans with the same cost choose the one with minimum completion time. There are no restrictions on the maximum plan cost or completion time.

Plan. For the problem described above, the planner presented in the next section finds the optimal plan shown in Figure 3. The plan has the cost 42 and completes at time point 85.

The earliest time component *First* can be executed is 12, because the required file *F1* needs to be transferred from *host2*. The data transfer takes at least 10

time units. The Network Service is asked to complete the transfer before the execution of *First*. Therefore NS has a slack of 2 time units. Similarly, component *Second* can start executing on *host4* at time point 35, and component *Third* is scheduled to run on *host1* at time point 65. No data transfers are scheduled for files *F2* and *F3*, because they are already available on *host1*. The final data transfer of file *F6* from *host1* to *host3* is scheduled without an upper bound, i.e. NS is asked to complete it as soon as possible.

4 Algorithm

The ACP has two important features. First, the size of the problem specification is often extremely large compared to the size of the plan. There are many hosts in the network containing a multitude of physical files, and various components available on the grid can run at different time slots on a variety of hosts. However, only a small fraction of this information is relevant to the problem. In addition, this information is not readily available for the planner, but rather the planner needs to explicitly ask external services for it.

The second issue is that many decisions need to be made in forward direction. For example, scheduling a data transfer requires knowing the location and the ready time of the source file, which implies that the part of the plan that produces the source file is completed before planning the data transfer.

Our planner performs search in the space of partial plans. Each partial plan is constructed using a combination of regression and progression techniques, which allows the planner to cope with the two issues mentioned above. Admissible estimates of the cost of completing partial plans are used to guide the search towards the optimal solution.

4.1 Basic algorithm

Our planning algorithm searches in the space of partial plans. A partial plan contains a set of nodes with a designated root node corresponding to the user goal, a set of flaws (a subset of plan nodes), and an admissible estimate of the plan cost. The main loop of the algorithm is shown on Figure 4.

The flaws of a partial plan include files that need to be obtained (by executing a component or reusing an existing file) and components that are not assigned hosts and time slots. Our current implementation first picks file flaws, and then places components starting from the leaves (i.e. components for which all required file nodes are already completely instantiated). Resolving flaws may result in creation of new partial plans, which are then put into the plan queue.

The flaws are resolved as follows.

File flaws are resolved by reusing a file already available in the network, reusing a component already present in the partial plan, or adding a new component (and all it required files) to the plan. For example, the file flaw *F6 on host3* is resolved by adding a new

```

Plan solve( File lfn, Host host ) {
    FileNode goal = new FileNode( lfn, host );
    Plan startPlan = new Plan( goal );
    planQueue = new Queue();
    planQueue.add( startPlan );

    while planQueue is not empty {
        plan = best plan in planQueue;
        if ( plan.flaws is empty )
            return plan;
        flaw = pickFlaw( plan );
        resolveFlaw( plan, flaw );
    }
    return FAILURE;
}

```

Figure 4: Main loop of the algorithm.

instance of component *Third* to the plan, which also requires creation of new flaws for files *F2*, *F3*, and *F5*.³ File *F2* is already present in the network and can be reused. The planner issues a query to RLS to locate physical replicas of *F2*. After an instance of component *First* is added to the plan to produce file *F4*, this instance can be reused to produce file *F3*. In some cases, it is beneficial to create a new instance of a component. For example, if transferring files *F3* and *F4* is expensive compared to adding a new instance of *First*, two instances of this component might be placed on nodes running components *Second* and *Third*.

To resolve a component flaw, the planner needs to choose a host and a time slot to run the component. The required information is obtained from the Match-Maker and Host Schedulers. File transfers are scheduled if necessary to deliver required files to the destination host by the beginning of the chosen time slot. In our example, the component *Second* can run on *host4* starting at time points 10, 25, and 35. However, only starting point 35 is feasible, because the required file *F4* is available on *host1* only at time point 22, and it takes at least 10 time units to transfer the file.

Because all file flaws are resolved first, the initial construction of plans is done using regression over the logical requirements of components. The actual resource allocation (resolution of component flaws) is done in the forward direction. This regression-progression combination is used for construction of each individual plan. The algorithm considers multiple alternatives using estimates of remaining plan cost to choose for expansion the most promising candidate plan.

4.2 Estimation of plan cost

Our algorithm relies on an admissible estimate of the plan cost to quickly guide the search towards the optimal solution. The cost of the plan is computed as sum of costs of all nodes constituting the plan.

³This information is obtained from the AR.

The costs of all file nodes, except the leaves, are zero. The costs of completely instantiated component and file transfer nodes are obtained from the corresponding services. The cost of uninstantiated component nodes (flaws) is approximated by the lower bound obtained from the Application Registry. Finally, the total cost of not scheduled data transfers is computed as follows.

For each path in the partial plan graph starting from the goal file node and ending in the first instantiated file node, estimate the cost and delay of the data transfer along that path as those for transferring the smallest file mentioned in the path between the hosts starting and ending the path. Estimate the cost and time of all uninstantiated data transfers as maximum over all such paths.⁴

4.3 Conflict detection

Several components of the same plan can be scheduled to run on the same host, and several data transfers can share a network path. This can cause resource conflicts and/or affect the completion time of the plan. To take such interactions into account, current reservations of a partial plan are supplied in calls to Host Schedulers and the Network Service. We have chosen this approach as opposed to post-processing the service responses, because the specialized algorithms used in those services might find a better solution than the necessarily pessimistic post-processing.

5 Future Work

The next two steps in working on the ACP with explicit reservations include evaluation of the model and improving the algorithm performance.

Experiments with real applications are required to evaluate adequacy of the model. The crucial aspect that affects design of the algorithm is the scale of the problem with respect to the number of answers returned by each of the services. Our model assumes that host and network reservation services can provide relatively high level of detail. Adjustments might be necessary to interface our planner with services currently available on the grid.

The model can be extended if necessary to provide additional functionality. For example, similar to data transfers, reasoning about disk space reservations on hosts might be added to the model.

Files required by a component can include not only data, but also code. This way our model can be extended to support run-time installation of software. In the simplest case this can be accommodated by a file transfer operation. In more complicated situations a special type of *software flaw* may be required. A procedure for resolving this type of flaw might take care of issues such as obtaining permissions and/or licenses, checking versions of required libraries, etc.

⁴This part of cost/time estimation is not yet implemented. Currently, the total cost and time of unscheduled data transfers is zero.

In general, the flaw selection and resolution methods can be extended to support other kinds of flaws. For example, in many scientific applications a parameter sweep needs to be performed. This operation involves calling the same executable multiple times with slightly different parameters. Planning for each of these executions separately might lead to poor performance of the planner. Creation of a special type of flaw resolved by a specialized algorithm can be beneficial.

The algorithm described above makes all reservations strictly in the forward direction. It is often the case that some intermediate step of the plan is highly constrained (e.g. there is only one host that can run a necessary component, and there are few time slots available on that host). Detecting such bottlenecks early can significantly prune the search. We plan to extend our algorithm with some techniques from the area of constraint satisfaction to take advantage of bottleneck resources.

6 Summary

In this paper we have described a model for the application configuration problem with explicit reservation of time slots. This work extends previous efforts in run-time construction of application configurations by supporting resource availability changing over time. We have also identified a set of services sufficient to provide the planner with required information about application components and network state.

To prove viability of our model, we have presented an algorithm for constructing optimal application configurations. Our algorithm takes into account both plan cost and completion time.

We have also outlined possible extensions of the model and modifications of the algorithm aiming to improve both expressiveness and performance of our approach.

7 Acknowledgments

This research was sponsored in part by NSF grants IIS-0097537, CAREER:CCR-9876128, and CCR-0312956; by DARPA agreements N66001-00-1-8920 and N66001-01-1-8929.

References

- Blythe, J.; Deelman, E.; Gil, Y.; Kesselman, C.; Agarwal, A.; Mehta, G.; and Vahi, K. 2003. The role of planning in grid computing. In *ICAPS*.
- Chervenak, A.; Deelman, E.; Foster, I.; Guy, L.; Hoschek, W.; Iamnitchi, . A.; Kesselman, C.; Kunszt, P.; Ripeanu, M.; Schwartzkopf, . B.; Stockinger, H.; Stockinger, K.; and Tierney, B. 2002. Giggie: A framework for constructing scalable replica location services. In *IEEE Supercomputing*.
- Do, M., and Kambhampati, S. 2003. Sapa: A scalable multi-objective metric temporal planner. *JAIR*.
- Fu, X., and Karamcheti, V. 2003. Planning for network-aware paths. In *Proc. of DAIS*.

- Golden, K.; Etzioni, O.; and Weld, D. 1996. Planning with execution and incomplete informations. Technical Report TR96-01-09, University of Washington.
- Kichkaylo, T.; Ivan, A.; and Karamcheti, V. 2003. Constrained component deployment in wide-area networks using AI planning techniques. In *IPDPS'03*.
- Nau, D.; Munoz-Avila, H.; Cao, Y.; Lotem, A.; and Mitchell, S. 2001. Total-order planning with partially ordered subtasks. In *IJCAI*.
- Paolucci, M.; Kawamura, T.; Payne, T.; and Sycara, K. 2002. Semantic matching of web services capabilities. In *ISWC*.
- Tangmunarunkit, H.; Decker, S.; and Kesselman, C. 2003. Ontology-based resource matching in the Grid - the Grid meets the Semantic Web. In *The First Workshop on Semantics in P2P and Grid Computing*.
- Thain, D.; Tannenbaum, T.; and Livny, M. 2002. Condor and the grid. In *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc.
- Wu, D.; Sirin, E.; Hendler, J.; Nau, D.; and Parsia, B. 2003. Automatic web services composition using SHOP2. In *Proc. of ICAPS'03 Workshop on Planning for Web Services*.