MeSch - An Approach to Resource Management in a Distributed Environment

Gerd Quecke [1], Wolfgang Ziegler [1]

[1] GMD - German National Research Center for Information Technology
SCAI - Institute for Scientific Computing and Algorithms
Sankt Augustin, Germany
{Gerd.Quecke, Wolfgang.Ziegler}@gmd.de

Abstract. Resource management in the typical Grid environment based on multi-MPP systems or clusters today still is one of the challenging problems. We will present MeSch, a solution for the problem of resource allocation and job scheduling in a distributed heterogeneous environment. MeSch has been implemented and tested successfully in the heterogeneous multi-MPP environment of GMD's Institute for Scientific Computing and Algorithms. MeSch allows users to access simultaneously, through a single request, heterogeneous resources distributed across the linked systems. This is possible either through explicit demands for different resources or through implicit scheduling of resources resulting from interpretation of requests. The scheduling system is available for both batch and interactive usage of resources. MeSch is implemented based on locally available scheduling facilities thus respecting the different scheduling systems and policies of the computing centers in the Grid.

1 Introduction

Resource management and job scheduling in the typical Grid environment based on multi-MPP systems or clusters today still is one of the challenging problems. Especially in a geographically distributed and heterogeneous environment it turns out, that although scheduling tools and policies are available for each subsystem, there is a lack of global resource management and thus, resource allocation is far away from being performed automatically. On the contrary: a substantial amount of human communication on all levels is necessary to partition the application, locate resources, and observe the behavior of distributed modules.

We will present MeSch, a light-weight solution for the problem of resource allocation and job scheduling in a distributed heterogeneous environment. The same way, a Grid application uses the Grid resources as a metacomputing environment allowing to make use of more than one MPP system or cluster, MeSch leads to the idea of building a metascheduler, which takes the burden of resource allocation for a metajob. The approach here is to build the metascheduler such, that it can use schedulers of all subsystems involved for all co-ordination and resource allocation tasks.

We will discuss the requirements, a local scheduler should have in order to be suitable for global scheduling. In addition, we describe the basic algorithm of a prototype MeSch metascheduler which allows co-ordination of the whole scheduling process during the application lifetime including resource allocation. The algorithm was especially designed to allow simultaneous access to the requested resources, a requirement typically needed by parallel applications.

2 State of the art

Until now, the only solution to overcome these problems is to use scheduling systems that are able to completely handle resource management for all resources involved. However, trying to use heterogeneous environments as they are becomes difficult if such attempts will be based on a single task approach as a regular service, without any need to change local administration rules and policies. Or, for example, to introduce local components like the GRAMs of the Globus [6] system building an additional encapsulating layer that interfaces to local resource management systems. However, this approach implies an “overhead” which may not be desirable for smaller computing centers.

We are well aware that there are other powerful systems like Globus, Legion or Unicore [6,7,5] providing a broader range of integrated tools. These projects are part of the foundations of the Grid and will be propagated more and more as the Grid evolves.

MeSch, however, is directed to a simple and efficient way of bundling distributed computing resources for the “bigger” parallel jobs of a user without the need to install one of the systems mentioned above.

3 Requirements for Global Resource Management

3.1 MeSch Scheduler Hierarchy

The MeSch approach handles resource allocation as a global task which can be divided into subtasks that may be delegated to co-operating schedulers of the subcomponents of a Grid environment. Ideally, we won’t discard local schedulers; instead, we build the metascheduler on top of the local ones. This allows us to build a hierarchy of schedulers.

In the same sense as a traditional scheduler maintains the nodes/processors as allocatable resources, the metascheduler does with systems (or partitions of systems). The advantage is, that all subsystems can act in their usual way with their own policy. Moreover, allocation of processors remains in the responsibility at the local system level and is not explicitly done by the metascheduler. As subsystems remain responsible for allocation, the local use of subsystems is not affected. No restriction is imposed on local scheduling strategies and administrative policies.

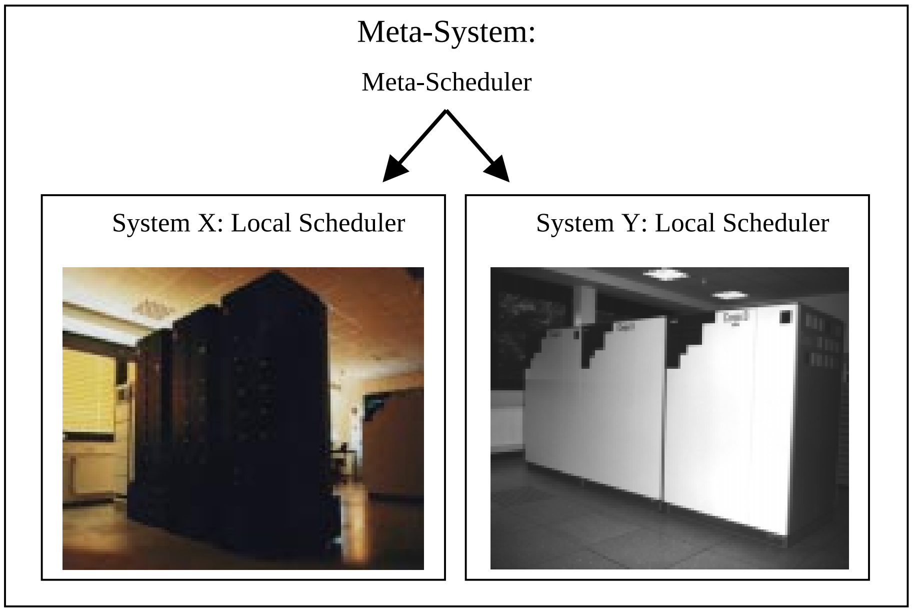

Fig. 1. Meta-Scheduler as a Hierarchy of Schedulers

The MeSch approach does not impose any restrictions on the type of Grid system: they may be homogeneous or heterogeneous, geographically distributed, any combination of MPP, cluster, and dedicated systems.

3.2 Requirements for Local Schedulers

However, MeSch requires some local scheduler attributes in order to be able to take over the burden of the overall scheduling task's global synchronization

In order to provide simultaneous access to required resources, methods of getting reliable information about suitable time slots must be available. This information enables MeSch to determine a common time slot on all Grid components that are required for a Grid application. First subscheduler suggestions about available time slots in general will not lead to a solution for the complete metajob. Thus, we must be able to ask for alternative time slots to have a chance to determine a commonly agreed time slot for simultaneous access.

If a commonly suitable time slot can be determined, the MeSch metascheduler must be able to inform each subscheduler to fix the time slot and to guarantee that it will allocate required resources at the agreed start time for the agreed time interval.

MeSch synchronization management requires several iterations of interaction with subschedulers to find a solution for a suitable time slot. Obviously, offered time slots must be (pre-)reserved by subschedulers, while they are under consideration for suitability. An allocation agreement protocol eases the synchronization process by defining a set of states a job may have from an scheduling request to its final execution.

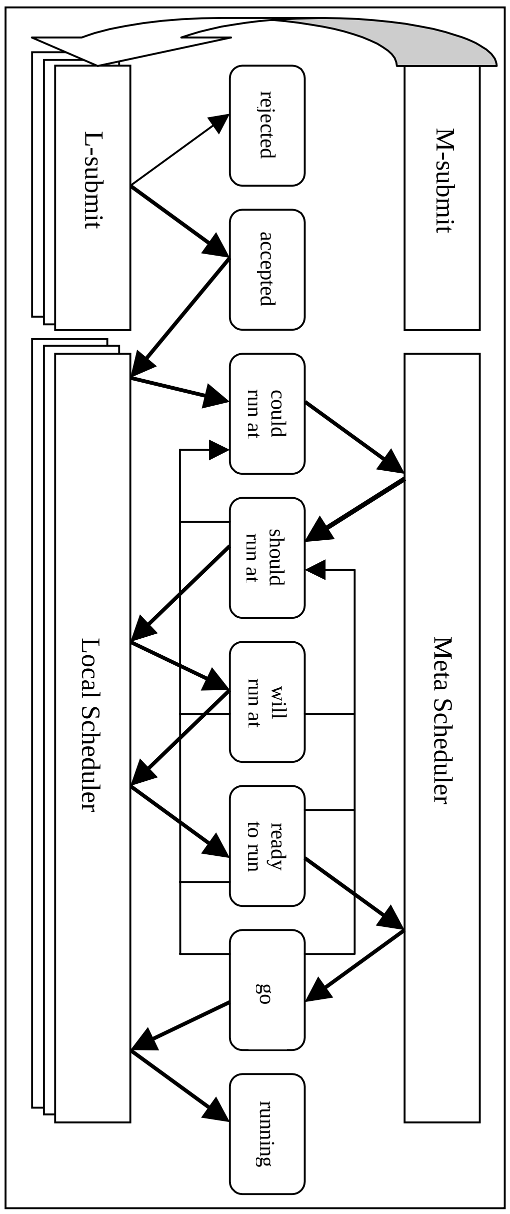

Fig. 2. Time Agreement Protocol

Each subscheduler, that can be modified to follow the allocation agreement protocol for metajob scheduling requests, is usable in a MeSch controlled Grid environment. Of course, if the local scheduler may not be modified to handle the allocation agreement protocol this may be implemented in a wrapper, given the local scheduler at least provides mandatory attributes like the ability to start jobs at a given time and to do estimation of future resource allocation.

3.3 Allocation Agreement Protocol

The goal of the allocation agreement protocol is to agree with all local schedulers in the allocation of local resources simultaneously for the same time interval.

We assume a specialized L-submit operation for each local scheduler, which accepts requests from the meta M-submit. L-submit knows that incoming requests are for a metajob, and thus it enables the time agreement protocol. This essentially means, that some additional information such as state, meta-identifier, etc. are to be maintained, and that the local scheduler knows about the time agreement protocol for this job.

M-submit calls the L-submit operation which sets the initial Accepted/Rejected state. A local preview will calculate a proposed CouldRun time, which the metascheduler will change to ShouldRun as a result of analyzing all meta components. The local scheduler will agree with a WillRun answer.

The ReadyToRun state, set by the local scheduler, indicates that local allocation is being prepared. The local resources are allocated, once the metascheduler has found common agreement by indicating the Go state.

If the analysis of offered time slots does not yield a solution, the metascheduler will go back in the protocol line and make better proposals. For the local schedulers: if they cannot fulfill a metascheduler request for a dedicated time slot, they make new proposals due to their local schedule policy.

4 A Prototype Implementation

4.1 Using EASY as a modeling tool

MeSch has been implemented and tested successfully in the heterogeneous multi-MPP environment of GMD's Institute for Scientific Computing and Algorithms. The available MPP environment allowed to attack the problem in a heterogeneous environment without having to deal with the problem of geographical distribution. Our prototype environment consists of an IBM SP2 with 34 nodes and NEC Cenju-4 with 64 processors.

Both systems use an enhanced EASY scheduler [1,2], which has been modified to fulfill the time agreement protocol.

The EASY concept is basically built on a “backfill” strategy. Our enhancements ensure, that even if a job has low precedence, it will be started, if its requirements do not have any implications for jobs of higher priority. The idea is to optimize system throughput by avoiding idle resources. As a side effect of this full backfill strategy, for all skipped jobs an estimated start time is available. This allowed us to provide a complete job estimation list, which is available to users and informs them about worst case start/stop times.

With this preview feature for each job, we have one basic property necessary to build MeSch: it enables MeSch to get actual information about scheduled time slots. In addition, an EASY job may be scheduled with a StartAt option.

For our prototype MeSch, EASY as a local scheduler fulfills the basic requirements for information gathering. The allocation algorithm had to be changed to support the time allocation algorithm. For the MeSch, in addition to new submit and release operations, only a previewer had to be implemented, which keeps control of the states of all local schedulers involved. The prototype allows submission of jobs in an EASY-like way.

5 An Example

In our prototype implementation, Meta components of the environment are specified explicitly by referencing the number of nodes of the respective systems, as in the example

```
msubmit –n70 –t300 –rSysA[30],SysB[40] -bmyjob
```

The batch job myjob requires 300 minutes of CPU time, 30 nodes of the SysA, as well as 40 nodes of SysB system. The command above will be separated by MeSch into the partial submissions

```
SysA: lsubmit –n30 –t300 –bmyjob
SysB: lsubmit –n40 –t300 –bmyjob
```

Each of the local systems will handle the respective request according to its local policy. Backfilling allows to find worst case time slot. According to the time agreement protocol MeSch is able to accept proposals from the local schedulers or to make new proposals until all schedulers involved agree to a common time slot.

As an example of the time agreement protocol, assume SysB accepting the metajob request. Refer to figure 2 for an overview of the state names and sequence.

With the EASY internal preview facility, the local scheduler of SysB signals a CouldRunAt time interval starting at time ts. The scheduler assures availability of requested resources to the metasystem. With the ts information from all local schedulers, the MeSch scheduler can determine the max(tsi) start time. A ShouldRunAt proposal to the local scheduler in general signals that for the reason of simultaneous access a later time than ts is favored by MeSch. A local scheduler may reject the request, proposing a new time slot and signalling the CouldRunAt state; it may accept the new time and signal a WillRunAt state for the respective part of the metajob. This iterative procedure leads to a commonly accepted WillRunAt state for each part of the metajob.

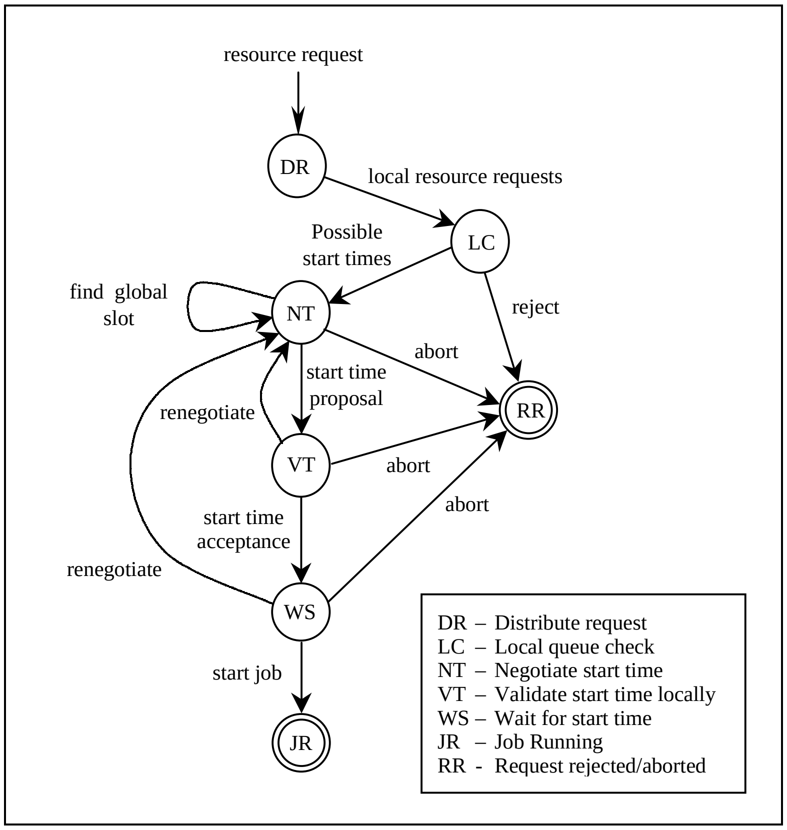

Fig. 3. MeSch Finite State Machine

At arrival of the agreed time slot, the local scheduler will signal a ReadyToRun to MeSch, which - if everything is right - will allow local scheduling to allocate the required resources immediately by signalling a Go state.

Each of these states may be rejected: Whenever the metascheduler is not able to accept a local scheduler proposal, it resets to the WillRunAt proposing an optimal time slot up to its knowledge about all local scheduler proposals. Whenever a local scheduler is not able to accept a MeSch proposal, it resets to the CouldRunAt state proposing the next available time slot for the metajob part, that can be guaranteed according to the local scheduling strategy and administrative policy.

6 Conclusion

The MeSch approach is a prototype metajob scheduler approach for a Grid environment. Its main advantage is that local scheduling policies are not affected by Grid jobs. The meta job scheduling can be viewed as using local schedulers as resource managers in a scheduler hierarchy. However, for an easy to implement allocation agreement protocol, local schedulers must provide a run time estimation facility for submitted jobs and accept and guarantee dedicated start time specification.

The practicability of the approach has been demonstrated by a prototype implementation based on an enhanced EASY scheduler version.

Currently we are investigating how to implement scheduling for visualization devices such as a workbench for applications with real-time visualization demand.

7 References

1. Lifka, D.: The ANL/IBM SP scheduling system. Job Scheduling Strategies for Parallel Processing, Lect. Notes Comput. Sci. 949 (1995) 295–303
2. Grund, H., Link, P., Quecke, G., Ziegler, W. EASY Job Scheduler for Cenju-3. Proc. of the First Cenju Workshop, HPC Asia '97, Seoul, Korea (1997) 20–34
3. Foster, I., Kesselman, C., (eds): The Grid: Blueprint for a Future Computing Infrastructure. Morgan-Kaufmann Publishers (1999)
4. Czajkowski, K., Foster, I., Kesselman, C., Martin, S., Smith, W., Tuecke, S.: A Resource Management Architecture for Metacomputing Systems. In Proc. of The 4th Workshop on Job Scheduling Strategies for Parallel Processing. LNCS, Vol. 1459, Springer Verlag, (1998), 62-82
5. Romberg, M.: The UNICORE architecture: Seamless access to distributed resources. In 8th IEEE International Symposium on High Performance Distributed Computing. LNCS, Vol. 949, Springer Verlag, (1999), 287-293
6. The Globus project: Globus Toolkit 1.1.3 System Administration Guide. (2000), http://www.globus.org/toolkit/documentation/globus_sag1.1.3.pdf
7. The Legion Project: Resource Management in Legion. (1998), http://legion.virginia.edu/papers/legionrm.pdf