

Resource Allocation on the BalticGrid

Kristaps Džonsons, KTH/PDC

kristaps@kth.se

Abstract

The BalticGrid has invested a considerable amount of time into the topic of resource allocation. In this document, we summarise the findings of this investment into a coherent plan for deploying a resource allocation strategy at the least cost and greatest benefit to the BalticGrid. Our plan involves the resource environment of the BalticGrid, current applications mapped to those resources, and the requirements set forth by the BalticGrid regarding the continued, service-guaranteed execution of these and other applications in a production environment.

1 Introduction

Resource allocation, regarding computation, is the allocation of compute resources in order to complete or partially-complete a job. A job, in this regard, is a stored compute routine; a resource is the corresponding memory, network, disc space, and so forth required for the computation's duration.

Resource allocation may concern the complete lifetime of a job or break it into chunks; it may schedule tasks by fixed reservation or by priority queuing; and so on. Allocation has been a central focus of computing since the inception of the batch computer and has been examined by some of the finest minds in mathematics and computer science. The volume of available literature on the subject is staggering.

In this document, we address the issue of resource allocation on the BalticGrid¹. We consider the following topics: what is the intended use of the Baltic Grid? What is the current and perceivable-future

pattern of access? The goal of this document is to determine whether the existing allocation strategy on the BalticGrid is acceptable given the current and future resource requirements of grid applications, and furthermore, whether this strategy implements all allocation requirements. We continue by questioning the usefulness of some requirements in light of the complexity of their implementation.

It's popularly considered that the current strategy of allocation does not satisfy the BalticGrid's requirements; furthermore, an experimental attempt to introduce a new allocator has met with failure. This document serves also to formalise the BalticGrid's requirements, thus showing whether the existing strategy is in fact under-performing and whether further experimentation is in fact required.

2 Strategy

The issue of allocation must be carefully approached due to the consequences of misapplication. A poorly-chosen allocator, whether in its theory or implementation, may result in unacceptable loss of resource efficiency in terms of both computation and man-hours. By following the strategy outlined in this document, we'll determine whether to consider a new strategy or continue maintaining the old, or possibly a mixture of the two.

The strategy proposed by this document is summarised as follows:

1. Determine the existing grid infrastructure viz. resource allocation and the technical environment of allocation.
2. Determine the resource requirements and expectations (trends and patterns) for existing and potential grid applications.

¹See <http://www.balticgrid.org>

3. Extrapolate common requirements of new, existing, and potential applications.
4. Perform cost-benefit analyses comparing maintenance of the existing strategy and implementation of a new one given the extrapolated requirements.

By following this strategy, we'll consider, in short, whether it's worthwhile to re-consider the allocation strategy, possibly abandoning existing strategy-bound applications, or whether the cost of changing is simply too great. Our consideration takes into account a variety of factors, with the greatest focus on grid middle-ware utilities. If applicable, we consider whether the costs of implementing a new system (or deploying another existing system) undercut the costs of maintaining the existing allocator for existing applications *and* the costs of fitting new applications to the existing allocator.

3 Existing Allocation Strategy

The notion of *middle-ware* is fundamental to the operation of the BalticGrid and thus its allocation requirements. Middle-ware is a term loosely defining an abstraction between a compute job and its operational resources. Middle-ware, thus, may serve to distribute jobs to their execution environments, assist in transferring resources to this environment, and correctly log the job's execution. A precise definition of middle-ware is beyond the scope of this document.

On the BalticGrid, jobs are executed upon submission to middle-ware, which subsequently invokes a resource allocator to acquire job resources. The only middle-ware on the BalticGrid is gLite² (using versions 3.0.0, 3.0.2, 3.1.0, and 3.1.1)[1]. Although other choices of middle-ware exist, they are not in production use.

3.1 gLite

In order to describe the BalticGrid's existing allocation strategy, it necessarily follows that we must

²See <http://glite.web.cern.ch/glite>

analyse the primary user, namely gLite. We preface our analysis of the BalticGrid's middle-ware, gLite, with a cautionary note on versioning. As of this document's creation, the BalticGrid uses versions 3.0.0, 3.0.2, and 3.1.0 of the gLite system. Documentation on the gLite system does little to address differences between versions; thus, our analysis may refer to obsolete or future versions, depending on the local gLite installation.

Furthermore, we underline that an exhaustive overview of the gLite system is far beyond the scope of this document. Here, we present a small view of the system as specifically applies to allocation.

In the gLite system, the management of compute resources is delegated to the JMS (Job Management Services) component of gLite [4], specifically by means of the WMS (Workload Management System) *to* the JSM (Job Submission and Monitoring)[5] with data *from* the ISM (Information Supermarket). Both of these services interoperate with the CE (Compute Element). Thus, the ISM provides information on available resources and the JSM schedules jobs on these resources. The WMS match-making service, whereby a job is assigned to a worker node, is the joint operation of the JSM and ISM service.

The CE must update the ISM with up-to-date information on available resources and accept job assignments from the JSM for processing. The means of interaction between the CE and WMS more or less defines the sequence of resource allocation within the gLite system. Using this terminology, the CE is equivalent to a resource allocator and notifier, which decomposes into the allocation of requested resources and the export of available resources. The exact sequence of this is defined in the gLite Architecture Document[5], §8.2-3, and the gLite Design Document[4], §6.2-4.

The CE mechanism in gLite is abstracted by the CREAM³ service, which provides a layer between the CE internals and calling components. This is a new addition to gLite from version 3.1. The CREAM interface then interacts with the underlying resource allocator, called the LRMS (Local Resource Management System). The LRMS performs the actual scheduling and execution of jobs

³See <http://grid.pd.infn.it/cream>

on the WN (Worker Node) as dictated by the JSM.

The BalticGrid's resource allocation strategy is thus limited by the LRMS and JSM gLite components. gLite assumes that both an LRMS and a JSM is already installed per site. The most common implementations are Torque and Maui⁴⁵, respectively. These are often bundled together as Torque/Maui[6]. Although Torque has its own resource management utilities, it's general design is for batch scheduling.

We note that our definition of "most common" is not rigorous, and involves the word-of-mouth of BalticGrid (and other grid institution) administrators. A canonical list of active LRMS and JSM implementations isn't available.

There are several alternative implementing services: SGE⁶ (Sun Grid Engine), LSF⁷ (Load Sharing Facility), Condor⁸, and others. There is experimental support for several more systems.[7]

The overwhelming BalticGrid scheduler/batch system combination is indeed Torque/Maui. Although other pairs have been used in experimentation, none of them are used in production. In this document, we continue to focus on Torque/Maui due to their ubiquitous deployment on the BalticGrid. We underline, again, that there is no canonical document detailing BalticGrid scheduler/batch system usage.

3.2 Torque/Maui

We thus follow our abridged analysis of gLite with an analysis of Torque/Maui. As with gLite, a detailed analysis is necessarily beyond the scope of this document.

These system have considerable pedigree: Torque inherits from the original PBS with modifications from many considerable institutions. Torque and Maui embody a full batch system: they function only to schedule jobs according to a granular job descriptor: a user specifies the minimum require-

ments of a job and the scheduler determines the optimal run-time. The conditions of scheduling are determined by static criteria.

Although the functionality of Torque is limited to scheduling, the focus of this document is, in fact, Maui, which implements the system of allocation and notification. Maui has a considerable number of features; in this document, we consider only those pertinent to the BalticGrid.

The Maui system operates by maintaining a table of resource availability on its service. This table is continuously updated by its WN components. This table may be translated and handed to the gLite ISM in order for gLite to match-make a job with resources. When a job is handed to the Maui system, it arrives with a JDL (Job Descriptor Language) file that enumerates the job's resource requirements. The Maui system subsequently schedules the job for operation on WNs by means of batch schedulers, usually Torque (but also SGE, LFS, and so on).

By abstracting its scheduling and allocation strategy, the gLite system is able to have a degree of independence from an existing cluster management system; the same is true for implementing a cluster system over an existing gLite deployment. Our analysis of the BalticGrid's resource allocation necessarily focusses on the default LRMS of its primary cluster application: the gLite middle-ware. In defaulting to the Torque/Maui system, the mapping of resources to jobs is limited to the functionality provided by batch semantics.

Potential resource allocators are thus limited to those supported by the gLite LRMS and JSM abstractions; or, for those not officially supported, at least providing significant support for interoperability with gLite systems. We note again that the interface exposed by the gLite system is limited to batch-like operations; a non-batch resource allocation system would have to be necessarily limited to this interface.

There is a further abstraction within the gLite system, the Grid Gate (GG), which acts as a resource scheduler for sets of clusters (such as the BalticGrid as included in the European Grid). At this time, we disregard this level of abstraction as it doesn't

⁴See <http://mauischeduler.sourceforge.net>

⁵See <http://www.clusterresources.com>

⁶See <http://gridengine.sunsource.net>

⁷See <http://www.platform.com>

⁸See <http://www.cs.wisc.edu/condor>

concern the BalticGrid’s internal resource scheduler; however, given the ubiquity of Torque/Maui as the JSM/LRMS for gLite sites, we assume that inter-operating this system with the GG is possible.

4 Resource Requirements

In this section, we discuss the resource requirements of the BalticGrid in terms of its running application. With this formalised, we’ll be able to accurately discuss whether the BalticGrid requirements laid forth in §5 are, in fact, reasonable.

The nature of the BalticGrid planned activity does not provide a stable, well-defined usage pattern for compute resources. Since activities deployed on the grid are necessarily experimental (as set forth in the BalticGrid’s mission statement). This produces erratic data in terms of operation and, more importantly, functionality of logging and accounting systems that produce source data.

Since little information is available, this section concentrates primarily upon a means of information discovery than summarising known information (as in §3, where most information already exists).

Since historically analysis of resource trends requires existing information, and source data for this analysis is often produced by a resource allocator, we’re necessarily limited to the information provided by the existing allocator, the default gLite CE, or inherit from other similar institutions. Since the BalticGrid isn’t intended to have “standard” operation during the current phase of deployment, we conclude that existing resource patterns are invalid and do not reflect future standard operational patterns.

A more stable approach to this problem is to consider current applications on the BalticGrid, instead of the usage patterns of these applications. Applications using the BalticGrid include GAMESS, DALTON, and analysis tools for the CMS and LHCb experiments. Although other applications run, or are intended to run, on the BalticGrid, we discount these in favour of the existing applications as representatives.

We note that there are many other applications, largely under the SIG (Special Interest Group) umbrella, which execute on the BalticGrid. However, in terms of compute hours, the following applications claim an overwhelming majority of available compute resources.

GAMESS⁹ is a quantum chemistry application. Rigorous measurements on the network and processor usage are not available, but informal measurements on the average run-time span upward from several hours, with an average run-time of several hours. A casual analysis of the underlying algorithms suggest high processor usage and a necessity for low-latency links between participating hosts. This analysis is speculative.

DALTON¹⁰ appears to have a similar running profile to the GAMESS system. This, too, is speculative.

Analysis of data from the CMS experiment, via the CMS¹¹, has a very well-defined run-time profile. Its algorithms, primarily Monte Carlo simulations, have a fairly simple parallelisation mechanism: they don’t require very high-speed interconnects. The CMS application does, however, require considerable disc space for its data sets.

The LHCb¹² application has similar run-time properties as the CMS experiment.

A casual analysis of these tools produces a broad set of resource requirements: DALTON and GAMESS require high-speed links, while CMS and LHCb require a great deal of disc space. All of these applications, we assume, require considerable memory and processor resources. The run-time duration of these is another significant commonality: all systems operate over significant amounts of time.

The similarity between these applications is in their duration and regularity of their operation. The loads are predictable, and in the case of unpredictability (as in informal reports of the GAMESS system), the unpredictability is on the range of hours of run-time. We can further assume, from the descriptions of these systems, that they consume a

⁹See <http://www.msg.chem.iastate.edu/games>

¹⁰See <http://www.kjemi.uio.no/software/dalton>

¹¹See <http://cms.cern.ch>

¹²See <http://lhcb.cern.ch>

maximum of resources at all times. The GAMESS and DALTON experiments favour high-speed links between networked computers, while the LHCb and CMS experiment don't. Both of these factors have a well-defined, predictable pattern of usage.

Until the BalticGrid releases detailed run-time statistics for its participating nodes, a study of execution profiles cannot be accurately determined. However, a reasonable speculation may be constructed from the profiles of the major relevant applications, all of which have a long history of usage in other computing environments.

5 Allocator Requirements

A full explanation of the BalticGrid's resource allocator requirements may be found on DJRA1.5[2] under the topic of QoS (Quality of Service) production. In short, this document states that the BalticGrid's requirements for a resource allocator are one that incorporates gLite and other potential middle-ware systems. This is a necessarily ambiguous definition. In practise, the allocator must satisfy these requirements while servicing the requirements of the existing BalticGrid applications as discussed in §4.

Since 2006, the BalticGrid has experimented in more sophisticated allocator features, such as advanced reservation and VO (Virtual Organisation) prioritisation. These experiments were conducted to added QoS (Quality of Service) guarantees to resource users. The system used in these experiments was Tycoon, discussed further in §6.1. Although no formal documentation exists that defines QoS requirements, we introduce a *post factum* summary:

1. Allow prioritisation based on job VO identity.
2. Allow fixed blocks of allocated time for "regular" jobs which don't wish to participate in the scheduling mechanism.

These considerably extend the requisite features for a BalticGrid allocation strategy, although they're general enough to be met by a variety of existing implementations discussed in the next section of

this document. A generalisation of these topics is to provide QoS guarantees to the existing service.

We stress that these requirements remain unformalised, which has led to considerable confusion regarding consistent requirement points. Nevertheless, we consider these to be compulsory requirements to the BalticGrid allocator that will considerable influence our analysis of allocation strategies.

6 Allocator Strategies

As stated in §5, a significant requirement of an allocator strategy is that it conform with gLite, whose design was described in §3.1. Those systems conforming to this requirement were also discussed in this section. In this section, we recapitulate our analysis of allocators and introduce the BalticGrid's experimental allocator.

A primary differentiator in the following sections regards static and dynamic allocation. A static allocator is one that accepts a table of resource availability and a job, then schedules the job for operation at the soonest-available and/or optimal slot. A dynamic scheduler, on the other hand, may preëempt and shuffle jobs in order to allow high-priority jobs to have lower latency. Both of these strategies have their merits.

6.1 Tycoon

Not all strategies need to inter-operate directly with the gLite system. The BalticGrid has experimented with a dynamic scheduler[8] (introduced in §5). Since gLite is limited to static scheduling, using a dynamic scheduler through the gLite interface is considerably difficult: there is no well-defined fashion of providing the ISM with a view of available resources when this view, in a dynamic system, may change at any time.

The BalticGrid extensively experimented with Tycoon¹³ in order to research the effects of dynamic scheduling, a thorough analysis of which is beyond the scope of this document. In short, the Tycoon

¹³See <http://tycoon.hpl.hp.com>

system creates a market environment where users are allocated credits. When a user wishes to acquire compute resources, she spends her credits in bidding in resources. In an environment with a surplus of demand, bids are awarded the highest bidder; thus, the user is encouraged to be frugal in spending and no prioritise jobs higher than necessary. A resource may continually accepts bids, at times descheduling existing executables in order to service a high-priority bid.

Since gLite doesn't, by virtue of the "Information Supermarket" and "Job Scheduler" design, interoperate well with dynamic allocation strategies, a continuous, coarse-grain Tycoon allocation was maintained for gLite, which was thus able to run its entire set of tools within a private network of Tycoon slots. Other middle-ware systems could follow in turn without modification; future deployments could work directly with the Tycoon system for dynamic scheduling.

This experiment had varying degrees of success and failure, and the official position of the BalticGrid is to discontinue its use^[3] of Tycoon due to poor implementation quality, questionable theoretic underpinnings and issues with availability. This doesn't discount, however, the underlying strategy of deploying a dynamic scheduler under a static, gLite interface. Furthermore, we allow that a credit-based scheduling system has superior benefits over a static system, if properly deployed; however, the architecture of gLite has poor design-level interoperation with dynamic systems.

Furthermore, another issue with a two-tier scheduling system is one of granularity. By allocating static slabs for un-modified middle-ware deployments (such as gLite), the interesting properties of a dynamic scheduler are considerably compromised. For example, if a gLite installation is allocated 50% of a host's resources, the allocator is never able to use this slab in making its dynamic allocations. This entirely defeats the purpose of a dynamic allocator.

This granularity is, however, a requirement of the BalticGrid, from §5, in terms of advanced reservation. We argue that the "advanced reservation" of Tycoon, however, is overly pessimistic in that a slab always exists for the gLite system in gen-

eral, and may not be further decomposed per gLite job. When no jobs are pending, or when only low-priority jobs are active on a gLite slab, there is no means of automatically narrowing the slab's resource availability.

Yet another issue is one of accounting. If a static slab is allocated per middle-ware system, jobs executed on that system are not accountable within the dynamic scheduler framework. Furthermore, jobs run on different middle-ware aren't routed, without an external agent, into a single auditing source. In short, there is a necessary duplication of work for scheduling, auditing and accounting within slabs. We reiterate that this defeats the purpose of a dynamic scheduler entirely.

In environments where the use of existing middle-ware systems is marginalised, a multi-tier scheduling system (static slabs within a dynamic system) may be preferable, as the primary use would be with the dynamic scheduling. However, the BalticGrid's requirements aren't so.

6.2 Torque/Maui

At present, the Torque/Maui system is the only production allocator in use on the BalticGrid. Our analysis in §3.2 suffices as an introduction to the system.

Beyond the usual facilities of allocation and scheduling, the Torque/Maui pair also support advanced reservation and prioritisation as dictated in §5. This feature is provided by ACL (Access Control Lists), which may be managed by BalticGrid administrators for the fine-grained access of resources per VO. This feature is extended with QoS services, wherein matching entities in the ACL are allocated resources not available to other nodes.

Both of these features, and more specific advanced reservation and QoS features, are available in the Maui administrator's manual.

7 Proposed Solution

We conclude our analysis with a proposal of allocation strategy in light of the BalticGrid's allocator

requirements (§5) and resource requirements (§4) by existing applications. We're conclusively able to state that the default allocator of gLite, if installed in a manner allowing for multiple middle-wares, is sufficient for the current needs of the BalticGrid.

The requirements set forth by the BalticGrid follow:

1. Full integration with the gLite middle-ware.
2. Ability to prioritise between VOs, among applications, and among users.
3. Ability to have both dedicated reservations and normal, queued operation on resource nodes.

Profiles of current applications follow:

1. Predictable, stable operational environments.
2. High dependence on processor and memory resources.
3. In limited cases, a high dependence on network usage.

In light of these profiles, the requirements appear fairly logical: there is no need, by these metrics, for preemption or other complex features.

Our choice of allocator settles on the existing system pair, Torque/Maui. These systems may be downloaded and installed separately from the gLite installation, with local gLite deployments using the installation for its resource management. Subsequent middle-ware systems, such as the Globus¹⁴ system, may also be configured as such.

This deployment differs from existing deployments in that usage of the QoS and advanced reservation features of Maui must be researched and documented, with administrators and users convening to discuss the resource requirements of particular VOs, or possibly specific applications or users within VOs.

With this scenario, there are two significant benefits to the BalticGrid. The first is that scheduling

and resource allocation for all participating middle-ware tools, and other tools, flows through the same system (Torque/Maui). The second is that auditing and logging may be configured in a similar manner (a discussion of auditing and logging is beyond the scope of this report). Although having a multi-tier dynamic and static scheduler is attractive, such as using Tycoon, the lack of inter-operation with gLite (and other batch middle-ware, such as Globus) is unacceptable. Furthermore, the current Tycoon implementation, in specifically considering Tycoon, has considerable problems.

In terms of cost/benefit, we find that using a Torque/Maui solution is acceptable for the BalticGrid. Furthermore, since Torque/Maui is the only allocation strategy currently running on BalticGrid, there's very little overhead in deployment. The only difficulty is that of researching the QoS and advanced reservation features and the necessary convening of administrators with users to plan a correct deployment schedule per VO.

References

- [1] BalticGrid Resource Usage. Online Publication, April 2008. <http://goc.grid.sinica.edu.tw/gstat/baltic/>.
- [2] BalticGrid. Account Management Productised: DJRA 1.5. Technical report, BalticGrid, February 2005.
- [3] BalticGrid. Final Experiences Report on Integration and Deployment of SLA and Account Management Components: DJRA 1.6. Technical report, BalticGrid, April 2008.
- [4] CERN. Design of the EGEE Middleware Grid Services. Technical report, JRA1: Middleware Engineering and Integration, August 2005. <https://edms.cern.ch/document/606574>.
- [5] CERN. EGEE Middleware Architecture. Technical report, JRA1: Middleware Engineering and Integration, July 2005. <https://edms.cern.ch/document/594698>.
- [6] CERN. EGEE User's Guide: WMS Service. Technical report, JRA1: Middleware Engineering and Integration, May 2006. <https://edms.cern.ch/document/572489/1>.

¹⁴See <http://www.globusconsortium.org>

- [7] Gergely Debreczeni. Generic Installation and Configuration Guide for gLite 3.1. Technical report, The EGEE Project, May 2007.
- [8] A. Payberah. Integration of User Level SLA for gLite Using Tycoon. Technical report, KTH, May 2007.