

The Gridkit Resource Management Framework

Geoff Coulson¹, Wei Cai¹, Paul Grace¹, Gordon Blair¹, Laurent Mathy¹, Wai Kit Yeung¹

¹Computing Department, Lancaster University, UK

[geoff, w.cai, gracep, gordon, laurent, yeungwk]@comp.lancs.ac.uk

A Grid execution environment consists of a number of interconnected computational nodes with contrasting capabilities. These must simultaneously support multiple distributed applications that must be mapped onto subsets of these nodes in such a way that their execution constraints are met. Furthermore, the resources required by each application vary over the applications' lifetime, as does the resource availability on each node. It is the function of Grid resource management middleware to appropriately map applications to nodes according to their resource requirements, and to dynamically manage the resourcing of applications as they execute.

Although successful, existing systems (e.g. Globus) have a number of limitations. In particular, they are coarse-grained in the sense that resource specifications tend to deal with whole machines (or at best processes) rather than with fine-grained resources such as threads, buffer pools, connections etc. In addition, they lack support for run-time adaptation - the resources allocated at application launch-time cannot be adjusted during runtime. Finally, existing systems don't offer any consistent notion of an *abstract* resource: they deal exclusively with concrete entities such as CPUs, memory bytes etc. This makes it difficult to map from application-centric notions of resource (e.g. "I need 3 matrix containers of type X, 1 buffer pool of type Y, a scheduler for EDF threads, and a Java virtual machine") to the notion of 'resource' that the system understands.

Our resource management design is realised as a (distributed) component-based resource management framework, which forms part of a larger Grid middleware known as GridKit. At the most abstract level, there are two parts to the framework: *i) global resource management* (i.e. coordinating resource management over multiple computational nodes) and *ii) local resource management* (i.e. managing resource allocation and usage in individual computational nodes). In addition, our framework operates over two distinct phases: *i) an initial resource allocation* phase, and *ii) a subsequent run-time resource management* phase that comprehends dynamic reconfiguration of resources in response to evolving application requirements, and in response to fluctuating resource availability in the infrastructure.

An application description which can be submitted to the resource management framework for execution consists of the following: *i) a set of top-level components that comprise the application*, *ii) a set of associations, or bindings, between interfaces and receptacles of these components that capture the abstract topology of the application*, *iii) a set of so-called tasks which, among other things, express the required QoS of different parts of the application*, and *iv) a mapping of tasks to components*.

The set of top-level components and bindings together comprise the compositional structure of the application. The bindings are annotated with QoS requirements; these annotations are used later when the various components of the application are mapped to physical computational nodes. The QoS ontology used is a pluggable element of the framework. In more detail, a QoS mapper function accepts plug-in *QoS mappers* which are defined on a per-application-domain basis. QoS mappers define a 'QoS ontology' that is meaningful for their associated domain, together with mappings from the ontology to a corresponding 'resource ontology'. For example, a domain of media transcoding applications might define QoS parameters such as "throughput in frames per second", "latency", and "acceptable frame degradation", together with mappings from these parameters to a resource ontology that comprehends concepts such as "buffer pool size", "number of high-priority threads" etc.

The ultimate goal of the framework is to appropriately place the application's constituent components on some specific set of physical computational nodes. It is the framework's job *i) to map components to nodes*, *ii) to ensure that each component's tasks are adequately resourced by its supporting nodes*, and *iii) to maintain the resourcing of the application at runtime as resource needs and resource provision fluctuate*.

This architecture promotes the integration of both fine-grained and coarse-grained resources to fully support end-to-end QoS guarantees. In our future work, we plan to further evaluate the functional and performance properties of the resource frameworks within a number of visualisation applications, particularly in the areas of forest fires and environmental informatics.