Architecture of an Intelligent Resource Management System

Omer F. Rana, David W. Walker and Yan Huang
Department of Computer Science
University of Wales, Cardiff
P.O.Box 916, Cardiff CF24 3XF, UK
{o.f.rana,david.w.walker,yan.huang}@cs.cf.ac.uk

Abstract

Implementing scientific or commercial applications on distributed heterogeneous computing systems, generally called 'Metacomputing', has recently become an active research area within the high performance computing community. This trend has been facilitated by specialised libraries such as MPI, and platform independent programming languages such as Java, which enable the aggregation of distributed CPUs, memory, storage, and data to support efficient concurrent execution. Similarly, Problem Solving Environments (PSEs) extend the notion of Metacomputing to also include the integration of scientific instruments and sensors, support for optimising the design of industrial processes and products, and for navigating and analysing large data sets.

At present, application developers must optimise execution performance of an application on a particular parallel machine, or on a cluster of workstations that constitutes a Metacomputing environment. Improving execution performance generally involves devising an effective application scheduling and allocation strategy. We describe an 'Intelligent Resource Management System' that is employed within a PSE for application scheduling, and one which can integrate third party software and provides support for computational steering. The resource management system can analyse trends in resource usage, and make recommendations to a scheduling system for task placement within a processor pool. This short paper describes on-going work within the Cardiff PSE project.

1 Introduction and Motivation

A Problem-Solving Environment (PSE) is a computing environment that provides all the resources needed to solve problems in a particular application domain [**?**]. Optimal scheduling of interacting tasks in a PSE, submitted by multiple users is a difficult problem, as the underlying hardware on which tasks are being run can vary significantly, and resource utilisation by different applications can also change in time and space. The scheduling task is difficult because: (1) Each heterogeneous computing resource can exhibit its own individual performance characteristics, (2) Computing resources can be shared, and under the control of different local schedulers and/or runtime systems, located in different administrative domains, and have different access rights to their utilisation. These access rights can vary with time, such as licensing requirements on particular software, or have prioritised access based on the domain from which access is requested.

The term 'computing resources' can also vary in granularity, from representing a single device, such as a workstation, a memory, or I/O unit, to a cluster of workstations or a complete domain (such as `cs.cf.ac.uk` – defined in terms of an IP sub mask). A global scheduling policy is generally difficult to develop in such a scenario, and integration with local scheduling schemes is preferred. From a pragmatic perspective, local system administrators for a parallel machine (for instance) are not always willing to relinquish control to a higher level resource manager, and licensing constraints can be such that some software can only run on a handful of machines.

This paper describes a resource management system for a PSE that can be integrated with third party schedulers, and can make recommendations based on application-specific resource usage. All resource usage is logged into a repository, which can be searched using an intelligent technique such as a neural network or a genetic algorithm. Resource requirements are defined in a special purpose 'Resource Specification Language' (RSL), that also enables the expression of constraints.

2 Intelligent Resource Management System

Our Intelligent Resource Management System (IRMS) maps an application task graph to a distributed and heterogeneous set of computing resources. The IRMS provides two main subsystems to support resource management, a subsystem that is application independent and provides support for heterogeneous resources and local resource management systems, and a subsystem that can account for application specific requirements. The IRMS architecture is illustrated in figure **??**, and must, (1) deal with resources that are owned and operated by different individuals or organisations, and which exist in different administrative domains. Resource usage may therefore have different security restrictions, or be subject to particular policy constraints, restricting scheduling of computation tasks to these devices (or groups thereof). Licensing constraints are a pertinent example of these, whereby mathematical or scientific software is restricted to a single machine or particular groups of machines; (2) deal with scenarios where a system administrator or a local runtime system will generally not grant permission to third party software to perform resource management on a given machine or within a cluster of machines. Hence an interface to a local run time system or resource manager (such as Codine etc) must be provided by the IRMS. Even the same runtime system may be configured differently at two sites, and hence the interface must be general enough to support local modifications for instance. Providing an interface to a third party resource manager may restrict performance, but will enable support for a wider range of networks and subsystems to be integrated. The alternative of providing a machine or cluster-specific manager will be initially investigated, with the higher level functionality explored in subsequent versions of the IRMS; (3) provide adaptability when application requirements and resource characteristics change during execution. This may occur when a choice must be made between running a simpler version of a component on a resource, versus not being able to run at all a more complex version specified by the user. The adaptability can be user supported, such as asking the user to choose a simpler version of a component when a more complex version would be restrictive, or it may be automatic, where alternative components in a task graph are selected, or the granularity of an existing component modified. This type of self-adaptability of an application graph must be constrained in time, and is supported by the performance model for each component.

These requirements are application independent, and form the core of the IRMS, and have been influenced by the Globus Resource and Allocation Manager [**?**]. The final objective is to forecast a schedule, which accounts for local resource managers, licensing and security constraints, dynamic resource information, and application specific constraints specified by the user. To deal with the last of these criteria, we also add an application layer around the IRMS services identified above, which enables every calculated schedule to be evaluated in terms of its impact on application performance. For instance, if data distribution between components of an application is frequent, then the network link between the components must not be the bottleneck. The application specific aspect of an IRMS must: (1) choose and filter resource combinations developed by the IRMS, based on constraints imposed by a specific application. This selection is supported by historical data (if available) on resource usage for the same application recorded during a previous run. If multiple possible schedules are identified by the IRMS, application specific constraints must be used to reduce or single out feasible resource configurations; (2) record resource usage for each subsystem within the task graph. The user is asked to identify individual components, or groups of components, whose performance must be recorded for later use. If a component of coarse granularity is to be used often for instance, then historical data on its usage may be used to constrain alternatives generated by the IRMS. Recorded performance data can be analysed in various ways, to look for patterns in resource utilisation within particular application domains, or for particular classes of components identified by users to be of relevance. Historical performance data may be shared by users. Two services within the IRMS are:

1. **The Information Service:** This service is required to gather and record usage data on selected resources within the PSE. Distributed resources can include both computing resources (processors, storage, visualisation equipment) and network resources. For computing resources this information includes operating system name and version, user name, host ID, number of CPUs and their type, speed, work load and usage, together with information on physical memory, virtual memory, swap space, and disk space. The information about network resources includes the network type, media, bandwidth, and average traffic. Each node participating in the PSE must support a local deamon

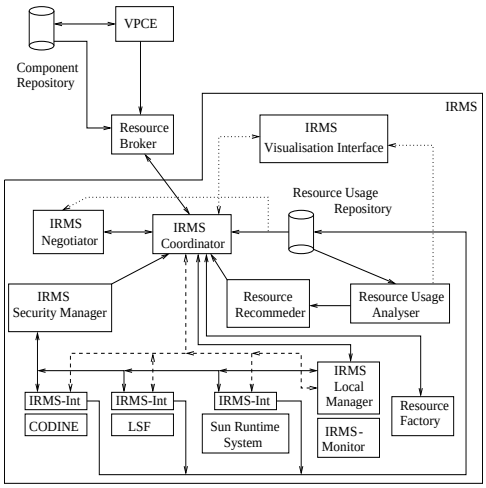

Figure 1: *The IRMS Architecture*

process, which is responsible for gathering the above usage data, and either recording it locally, or transmitting it to a central usage repository. If computing resources are being managed by a third party system, then the IRMS links into data structures within these systems, to enable tracking of jobs.

2. **The Scheduling Service:** This service includes a Scheduling Policy Service, a Scheduling Definition Service, a Scheduling Broker Service, and a Mapping Service. The Scheduling Police Service provides multiple, site-specific scheduling policies such as earliest deadline first. In addition, intelligent scheduling policies, based on methods such as genetic algorithms and neural networks, may also be used. The scheduling definition services provides a description language based on Globus RSL [**?**] to enable users to define their own scheduling constraints within the PSE. The Scheduling Broker Service selects a policy based on precedence constraints identified by the generated task graph, and input from the resource information service. The Resource Broker communicates with the IRMS Coordinator, with the latter acting as the central controller for all operations undertaken by the resource manager. The coordinator must ensure the integrity of all other components in the IRMS, and provides an external interface to other components within the PSE utilising the IRMS. When local resource managers are not available, the IRMS Coordinator interacts with an IRMS Local Manager for resource allocation within a given domain. Resource descriptions are undertaken using a set of predefined tags. For instance, to describe two workstations running Solaris:

```
<resource group id=Solaris>
 <resource id=celerity>
  <hostid> celerity.cs.cf.ac.uk </hostid>
  <workload> 100 </workload>
   <connectivity>
    <medium> ethernet </medium>
    <port> udpA </port>
   <connectivity>
 </resource>
 <resource id=pallin.disk1>
   <speed> 7500 </speed>
   <size> 2.6 </size>
   <connectivity> IDE </connectivity>
 </resource>
</resource group>
```

which describes a workstation called `celerity` which can run a maximum of 100 jobs (the job granularity is described elsewhere in the description) is connected over an Ethernet, and can receive UDP packets. Similarly, the local disk on a workstation `pallin` can be referenced as `pallin.disk1`.

A Resource Usage Repository records execution traces on various resources that form part of the PSE. The time between snap shots of each subsystem can be specified by the user, and is determined by how much information must be recorded, and the types of information made available by third party resource managers such as Codine and LSF. All information is recorded using the same tags as used in the example above. In networked resources, periodic measurements of latency and/or bandwidth across a link may be necessary, for instance. In systems where local resource managers are not active, an IRMS Monitor is used as a sensor to record device specific information, and works in conjunction with the Local Resource Manager for scheduling and allocation of user tasks. The user must identify which resource parameters to monitor, with a range of default parameters specified for resource types, such as processors, disk, memory, network link etc.

Other services include a **Resource Usage Analyser** for deriving patterns from resource usage for particular components, a **Resource Recommender** module to identify resources best suited to given components. An IRMS **Security Manager** acts as a gatekeeper to perform authentication and verify job transfer between resources. An IRMS **Negotiator** serves a crucial role in adapting application requirements to resource availability. On receiving a request from the Resource Broker, the IRMS-Coordinator checks each component identifier with the Resource Usage Repository to determine previous component usage. If a match is found, the previous usage information can be used to determine a suitable schedule. The Resource Recommender is queried for possible alternatives, and a suitable alternative is chosen based on an optimisation over all components within the application. If the component has not been used before, then it is mapped to the first available resource by the IRMS-Coordinator, and using a local resource manager dispatched to the resource(s). If an allocation cannot be made, the IRMS Negotiator is invoked to ask the user for alternatives. Allocation is therefore performed in order of the task graph and any dependency information obtained from the VPCE and the Resource Broker.

3. **Implementation:** The IRMS is implemented using the TAO ORB. TAO provides end-to-end Quality of Service guarantees, such as end-to-end priority preservation, hard upper bounds on latency and jitter, and bandwidth guarantees. Each service described above is implemented as a separate CORBA object, and has a pre-defined interface defined in CORBA IDL.

3 Conclusion

An architecture for an intelligent resource management system, for scheduling applications within a distributed, heterogeneous environment is described. The IRMS is currently being developed as part of the Problem Solving Environments project at Cardiff, and enables the integration of third party software, analyses patterns in resource usage, and makes recommendations to a user. The TAO ORB is being used to implement the system, because of its support for multi-threading and a low latency communication mechanism.

The IRMS offers three novel approaches to resource management in Metacomputing environments: (1) the ability to integrate third party scheduling services, to enable scheduling policies to be decided locally, and overcome the problem of systems administrators having to surrender control to a higher authority, (2) analysing resource usage data to derive patterns of usage within an application, and using a Negotiator and Recommender to suggest possible scheduling schemes that have been successful in the past, (3) to provide a resource description language that can be shared as text documents, and one which enables constraints to be specified using specialised tags.

References

[1] The GLOBUS project. http://www.globus.org/.

[2] J. R. Rice and R.F. Boisvert. From Scientific Software Libraries to Problem-Solving Environments. *IEEE Computational Science and Engineering*, 3(3):44–53, 1996.