Agent based Resource Discovery for Dynamic Clusters

Omer F. Rana, Daniel Bunford-Jones
University of Wales, Cardiff
o.f.rana@cs.cf.ac.uk

Matthew Addis and Mike Surridge
IT Innovations
mja@it-innovation.soton.ac.uk

David W. Walker
Oak Ridge National Laboratory
walker@msr.epm.ornl.gov

Ken Hawick
University of Wales, Bangor
khawick@sees.bangor.ac.uk

The resource management problem in the simplest form consists of, (1) selecting a set of resources on which to execute jobs generated from an application, (2) mapping tasks to computational resources, (3) feeding data to these computations, and (4) ensuring that task and data dependencies between executing tasks are maintained. In current systems, resource selection or discovery generally involves identifying suitable computational engines from a pool, typically homogeneous, based on criteria ranging from licensing constraints to processor(s) capability(ies) and background workload. A good overview of such cluster management systems can be found in [1]. A MatchMaker (MM) based approach is described, which can be used in collaboration with existing metacomputing systems such as Globus, and infrastructure APIs such as Jini. The MM acts as a “yellow page” service to enable tasks to find resources for execution. The MM approach can support dynamic clusters, i.e. resources can enter and join their local clusters at will. The MM is therefore a higher level service that can provided additional support to existing resource management services, and a testbed to enable machine learning approaches to be considered in the context of resource management for dynamic clusters. The scope of the approach is also very general, and can be extended to deal with mobile devices in addition to devices with fixed IP addresses.

Each member of the resource set $\mathcal{R}(t)$ is wrapped as an object, with the availability of the object at time $t$ accessible via an interface, and methods identifying services that can be invoked on the object by a user. The availability over time extends from $t < t_{current}$ (usage history) to $t > t_{current}$ (projected usage) and includes $t = t_{current}$ (current usage). Availability over time is just one of the parameters supported in the system, for instance, we also consider availability over the set of resource users. The matchmaking service works as follows: Each $R_j$ sends an asynchronous message to a pre-defined matchmaking service ‘M’ (running on a given host) to indicate its availability within a cluster. Each message is tagged with the resource type: (1) computational resource ‘C’, (2) data storage resource ‘S’, (3) visualisation resource ‘V’, or (4) scientific instrument ‘I’. The message contains no other information, and is sent to the local ‘M’. On receiving the message, the local ‘M’ responds by sending a document specifying the required information to be completed by the resource manager at $R_j$. This information is encoded in an XML document, and contains specialised keywords that correspond to dynamic information that must be recorded for every device in the pool. The form also contains a time stamp indicating when it was issued, and an IP address for ‘M’. The form can be automatically completed using scripts, or it can be completed manually by a systems administrator. The manager for $R_j$ completes the document, and sends it back to ‘M’, maintaining a local copy. The document contains the original time stamp of ‘M’, and a new time stamp generated by $R_j$. Some parts of the document are static, while others can be dynamically updated. Once this has been achieved, the new device is now registered with the resource manager, and will continue to be a suitable candidate for task allocation until it de-registers with ‘M’. If the device comes off-line or crashes, ‘M’ will automatically de-register it when it tries to retrieve a new copy of the document. We define each resource document as $\mathcal{D}_R$. A user wishing to execute an application issues a request document based on requirements of each task $T_i$ within the application, and classed using the ‘C’, ‘V’, ‘S’, ‘I’ annotation to the matchmaking service ‘M’ within the local cluster. This results in a set of documents being sent to
the user, for each $T_i$ in the application. The user has complete control over the granularity of $T_i$, and tasks may be grouped based on known dependencies. Each document must now be completed by the user, either using pre-defined scripts or manually. The issued document contains a time stamp, and on subsequent return to ‘M’, contains a time stamp from the user. We define each task document as $D_T$. ‘M’ now tries to find a match between each $D_R$ and $D_T$, based on pre-defined matchmaking criteria. Each time a suitable match is achieved, ‘M’ sends the generator of $D_T$, and $D_R$, their corresponding identities. The matched participants must now activate a separate protocol to complete the allocation, and this process does not involve ‘M’. In this way our matchmaking approach works as a resource discovery and recommendation technique and does not replace the resource management functionality of $R_j$. After the application user and resource provider have engaged to establish and confirm a concrete reservation or allocation of the resource to the task, the matched $R_j$ must now either de-register itself, or submit a revised document $D_{R_j}$ that describes its change in availability. If a local ‘M’ within a cluster cannot fulfill a request based on the submitted $D_R$, then it can forward this request to an ‘M’ within another cluster. The matchmaking services are therefore federated, and register with each other using a pre-defined document $D_M$, which identifies their IP address and start time. A prototype system has been implemented using the JKQML [2] libraries.

In the MatchMaking process first the set of resources is identified which are capable of performing the given task. Then the list of capable resources is refined by identifying the subset of resources that are also available for the task, for example the ones that the user has both the permissions to use and are available time-wise. The complexity of the match can vary, and involves three approaches (adopted from RETSINA/Larks [3]): (1) Syntax Match: Tags in both documents are compared to find exact matches based on common syntax. (2) Context Match: “context” is the distance between the keyword identified by the user, and a similar keyword present in the device ontology. Terms are connected using ‘isa’ or ‘hasa’ relationships, where the former expresses a specialisation, and the latter an attribute. (3) Semantic Match: This type of matching requires finer detail about the keywords in the XML document, and their associations. In the RETSINA work, this is achieved using an associative network, where terms are connected to each other using a network based description, with the weight on the arc connecting concepts representing the degree of similarity between two terms. Having determined the set of capable resources, the task requirements and user permissions are compared with resource availability. In the simplest form, this would involve comparing user deadlines for task completion against temporal availability, and by checking that the user has the permissions to use a resource (user availability).

It is important to note that the MM services does not replace a scheduler/dispatcher, but acts primarily to find suitable resources for given tasks. Once one or more matches have been made, the MM informs the participants involved, and deletes the task specification documents. It is now up to the participants to invoke an allocation protocol to achieve task mapping to the matched resource. Once the mapping has been finalised and the task been either reserved or dispatched to the resource, then the MM can request new availability information.

References

