Performance-sensitive Service Provision in Active Digital Libraries

Georgousopoulos Christos  
*University of Wales, Cardiff (UK)*  
geolos@cs.cf.ac.uk

Omer F. Rana  
*University of Wales, Cardiff (UK)*  
o.f.rana@cs.cf.ac.uk

**Abstract**

An agent-based architecture of an active Digital Library (DL) is first described, to illustrate how electronic service provision can be supported through the use of agents. The use of mobile agents is presented as a key enabler for allowing services to be combined from a variety of providers. Load balancing approaches are then used to illustrate how particular performance criteria can be achieved in service provision. Extrapolation of the approach to the general Service-Oriented computing model is briefly discussed.

1. **Introduction**

Digital Libraries (DL) provide a useful way to group a collection of services and digital objects. Recent focus on active DLs, whereby content from a collection of different repositories may be aggregated, provides useful parallels with work in Service-Oriented computing. Such repositories can be implemented using specialized hardware, and often support domain-specific interfaces. Accessing the content available within such a DL through the use of intelligent agents provides an important step in re-purposing such DL content. An agent, in our work, is seen as software capable of conforming to FIPA standards\(^1\), and having a very specific functionality (role) within a larger system. Interaction between such agents is then based on demands made on them by other agents. In this way, a collection of agents can be used to provide a service-based layer to access the contents of a DL in a variety of different ways.

2. **Load balancing**

Generally, load balancing aims to improve the average utilization and performance of tasks on available servers, whilst observing particular constraints on task execution order. Assuming agents have a set of tasks to execute, it is necessary to identify how these tasks may be distributed across available servers. Hence, workload distribution must consider both the number of agents on a server and the number of tasks being executed by each agent. Load balancing can be either static or dynamic according to the multi-agent system in which it is being considered. In static load balancing tasks cannot be migrated elsewhere once they have been launched on a specified server. In dynamic load balancing a task may migrate to another server, utilizing the agent’s mobility. Keren and Barak [2] show that dynamic load balancing outperforms the static case, with a 30-40% improvement over the static placement scheme.

There are two basic approaches to distribute tasks among servers: the state-based and the model-based approach. In the state-based approach, information about the system state is used to determine where to start a task. The quality of this decision depends on the amount of the state data available. Gathering the data is expensive, but leads to a more accurate decision. In the model-based approach, load balancing depends on a model which predicts the system state and which may be inaccurate. Model-based approaches are more difficult to implement as they involve the derivation of an initial model, and the need to adapt the model over time. In state-based load balancing, a common approach for managing system state and load is the market mechanism to value resources and achieve an efficient match of supply and demand for resources. Examples include Spawn [3] using a negotiated auction protocol, and OCEAN [5] based on non-negotiable pricing mechanisms. System state may be accumulated in different ways, via specialized monitoring agents, such as Traveler [4]. In FLASH [7], a system agent maintains information about the whole system state and passes it to node agents on each server in the network. Node agents monitor locally residing mobile agents. User agents (which are mobile) are responsible for the load balancing of the parallel application, and migrate

---

\(^1\) FIPA: [http://www.fipa.org/](http://www.fipa.org/)
through a cluster searching for free resources. Their migration decisions are based on internal states as well as internal and external events. Almost all the systems that explore the model-based approach use distribution of CPU load and expected process/task lifetime to decide if and when to migrate. Malone’s Enterprise [6] uses a market mechanism, and Challenger [1] uses a learning-based approach. Most of these approaches however cannot easily adapt to changing system workloads.

2.1. Model-based load balancing

The model-based approach uses CPU utilization and emphasis is given to the prediction of task lifetime. The model adapts over time due to the information gathered from the state-based approach. Reliable capture of system state is important, and therefore the information exchanged between the management agents (MAs) is an important factor for prediction. The main information exchanged is the number of agents on each server and the number of available servers along with their utilisation indexes. Irrespective of the distribution algorithm, a common policy is that a task should be assigned to the least loaded server – assuming that servers are of equal processing power. Consequently, the more accurate the estimation of a server’s utilization, the better the load balancing decision.

The utilization of a server at any point of time is directly correlated with its load i.e. the tasks being executed at that time. Malone [6] defines the utilization of a system as: 

\[ U = \frac{\alpha \cdot \mu}{L} \]

where, \( \alpha \) is the average number of job arrivals per time unit, \( \mu \) is the average job length, and \( L \) is the total processing power in the system. Therefore for a given server, \( \alpha \) corresponds to the number of agents on that server (assuming that there is a one task per agent), \( \mu \) to the average task time of the \( \alpha \) agents, and \( L \) to the total processing power of the hosting server. A server’s utilization in relation to its processing power \( L \) can be estimated. Such a comparison on utilization values helps identify a server that will be unloaded first i.e. will accomplish all of its tasks first. Accuracy of estimating a server’s utilization is based on a perfect estimation of the agent task lifetimes. The more accurate the average task time \( \mu \) of agents \( a \), the more reliable the corresponding server’s utilization. The actual utilization of a server can be measured by using specialized routines/utilities (like xload or ps, available on the Unix OS) that provide the percentage CPU usage. As CPU usage changes frequently, LB decisions should not only rely on the current utilization but on future load predictions (hence Malone’s formula is more useful than system tools).

2.2. Model and Adaptability

Load balancing decisions are based on a model which accepts as input an agent’s requirements and the system state information, and gives as output the appropriate server(s) where the particular agent should migrate to in order to fulfill its task. The model is a function of the: (1) agents’ tasks, (2) servers’ utilization (workload), (3) availability of resources at the server, and (4) network efficiency. The itinerary of an agent is constructed by its local MA each time before the initiation of its task (which may be simple or complex). The itinerary of an agent with a simple task comprises a list of server addresses, with the appropriate resources in descending order, based on the servers’ workload which can serve the agent’s task. The first server on the list is characterised as the ideal one where the agent can accomplish its task fastest, and the rest provide alternative option of migration. The construction of an agent’s itinerary may require input from the MA twice. Initially, an itinerary composed of suitable servers is created for the acquisition of the appropriate data (a simple agent task for instance), subsequently a second itinerary for the processing of the data (after they have been collected) consists of a list of compute servers. The existence of two separate itineraries is compulsory – as it is impossible to decide where a filtering task can be performed, as the amount and kind of data to be processed is unknown. Secondly, as resource conditions change frequently, decisions on load balancing must be taken directly before the initiation of a task.

It is sometimes impossible to estimate task lifetimes beforehand (e.g. the time a user is running a remote application) or such estimates may be erroneous. To deal with such errors in estimation an estimation error tolerance parameter may be needed. If a task takes significantly longer than it was estimated to take (i.e. more than the estimation error tolerance), the server running the task aborts it, and notifies the user who initiated the task that that task has been cutoff. This cutoff feature prevents the possibility of a few users or tasks monopolizing an entire system. In our model for simple agent tasks, the lifetime is predicted to be equal to the average task completion time in previous runs. For complex tasks, lifetime can be estimated based on calculations on the collected data to be filtered. The major parameter on distributing tasks among the servers is the workload on the available servers. If the lifetime of complex tasks was unknown, then the model would not function properly, since complex tasks influence the utilization of a server significantly more than simple tasks. To be more generally applicable, the
model should provide a means of self-adapting to error estimations. The algorithm is activated by the MAs and its objective is to monitor the utilization of every server and amends it when it is miscalculated, due to the introduction of agent tasks with unknown lifetimes.

The utilization of a given server impacts average task completion time on that server. Furthermore, this utilization changes when its agent load changes i.e. on agent entry/departure. As lifetime of complex tasks is unknown, the selection of servers is based on the current utilization and not on their predicted utilization. Hence, utilization on the arrival of a complex task is not actually affected, resulting in an incorrect evaluation of a server’s utilization. Our algorithm runs on each server separately, and on arrival of the first agent sets a timer, and calls check_AvTaskComplTime. Initially the timer is set equal to the average task completion time of agents on the corresponding server. If no agent has left the server until check_AvTaskComplTime is called, this implies that the agents present on the server (or even the first agent that arrived on the server) have not accomplished their task by the expected time i.e. the time corresponding to the average task completion time of an agent. The average task completion time of agents is updated only after the departure of an agent from that server. Therefore, until an agent accomplishes its task the utilization is unchanged. The algorithm’s objective is to update the utilization of that server based on the increase in the average task completion time of those agents that have completed, and publish this information to the rest of the MAs. This updated must be repeated periodically. The timer activates the check_AvTaskComplTime procedure due to the change in the average task completion time of agents. It is stopped when there are no agents left on the server.

3. Experimental results

Figure 3 displays the utilization of each of the five information-servers used in our DL prototype during the launch of 200 agents. The load balancing experiments were performed on a 100Mbit/s Ethernet with 6 Sun-ultra 5 workstations running Solaris 8 and Voyager 4.5 agent platform from Recursion Software. Of the available machines, five were used as information-servers and one as a Web-server. Every information-server had a data repository maintained by the Oracle DBMS, composed of replicated test-data. Each server had identical computational capabilities. The introduction of agents with complex tasks in the agent load resulted in higher deviations of a server’s utilization. The second graph of figure 3 illustrates the utilization of the information-servers on which 15% of the agents launched had complex tasks.

The variations in utilization of each server are higher in comparison with the previous graph, due to the arrival/departure of agents with complex tasks that require more time to be processed. Though it can be observed that the utilization of each server fluctuates at the same level as the other servers, where after a high drop in utilization of a server caused by the completion of one or more complex task(s), there is a rise to set the server’s utilization equivalent to the utilization of the other servers.

3.1. Adaptability of Load Balancing Algorithm

To explore algorithm adaptability, three different schemes are used; labeled LB No.1, 2 and 3 in figures 4, 5, and 6. Scheme 1 represents the default load approach in our DL system. In scheme 2 the lifetime of complex task is unknown. In scheme 3 the adaptable algorithm is utilized for amending the server’s utilisation after introduction of agent tasks with unknown lifetimes. The experiments are used to determine when scheme 3 reaches the performance of scheme 1; in other words, test the functionality of the adaptability algorithm by systems within which the lifetime of complex tasks cannot be estimated or
predicted successfully. The performance of each load balancing scheme for 200 agent tasks among five information-servers, is presented in figure 4. All tests are based on a variable introduction of tasks to test the efficiency of the algorithm.

![Figure 4: Total task time required by agents to complete their tasks](image)

Figure 4: Total task time required by agents to complete their tasks.

Lower values in Figure 4 indicate lower time required by the agents to complete their tasks in total, thereby resulting in better load balancing. Each output is based on results from four experiments on each of the three load-balancing schemes for six different variable introductions of tasks, resulting in the launch of 14,000 mobile agents in total. As scheme 1 is based on known or correctly predicted lifetime of agent tasks, load is correctly balanced amongst each server; where obviously when there are no complex tasks involved all of the three LB schemes behave the same. The difference in performance between schemes 1, 2 and 3 is expressed in figure 5. The difference in performance between schemes 2 and 3 is due to the utilization of the adaptability algorithm. Figure 6 reveals the performance improvement arising from the use of the adaptability algorithm.

![Figure 5: Efficiency between LB scheme No.2 and No.3](image)

Figure 5: Efficiency between LB scheme No.2 and No.3.

It can therefore be inferred that the higher the introduction of complex tasks of unknown lifetime in a system (from 5% to 25%), the better the load balancing by the use of the adaptability algorithm – with an improvement of between 1.63% to 10.8%.

![Figure 6: Optimization of LB scheme No.2, based on the utilisation of the adaptability algorithm](image)

Figure 6: Optimization of LB scheme No.2, based on the utilisation of the adaptability algorithm.

4. Conclusion
The design of a load balancing model depends on the properties and functional needs of the agent-based system. The proposed model may be employed by other systems utilizing active archives, in which the lifetime of complex tasks cannot be estimated or tend to be erroneous. System developers can take advantage of the adaptability of the model to cater for variable system workloads.

References