Combining State and Model-based Approaches for Mobile Agent Load Balancing

Christos Georgousopoulos
Department of Computer Science
Cardiff University, P.O. Box 916
Cardiff CF24 3XF, UK
Tel: +44 (0) 7799417703
gelos@cs.cf.ac.uk

Omer F. Rana
Department of Computer Science
Cardiff University, P.O. Box 916
Cardiff CF24 3XF, UK
Tel: +44 (0) 29 2087 5542
o.f.rana@cs.cf.ac.uk

ABSTRACT
An approach for dynamic load balancing of mobile agents is described. We demonstrate this approach for a multi-agent system operating on an active digital library composed of multi-spectral images of the Earth, as part of the Synthetic Aperture Radar Atlas (SARA)[25]. In the proposed architecture specialized stationary agents are used to gather system state information and make decisions on the distribution of mobile agents among the servers. Our approach is based on a combination of a state and model-based approaches to load balancing.

Keywords
Load balance, mobile agents, digital libraries.

1. INTRODUCTION
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, although there may be an even distribution of agents among the servers, the load on the servers is not balanced due to the different amounts of work undertaken by each agent.

Load balancing can be either static or mobile[10] 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 mobile load balancing a task may migrate to another server, utilizing the agent’s mobility. Keren and Barak[14] show that mobile 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 to load balancing are much rarer, as they involve the derivation of an initial model, and the need to adapt the model over time. Little work exists on integrating the state and model-based approaches for load balancing i.e. to construct and adapt the model with minimal state information. This is the focus of the work presented here.

1.1 State-based Load Balancing
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. Spawn[22] was the first system to employ market-based load-balancing strategies. Some systems use only a price, and match offers and bids, while others employ more sophisticated auction protocols[17], such as Vickrey auction, sealed-bid double auction, repeated clearing-house double auction etc. In these approaches consumers bid for resources according to the auction style being employed. An advantage of auctions as a market mechanism is that they allow one to determine an unknown resource value in a group of agents. However, this agreement comes at a price; the communication needed to determine it. An alternative to auction protocols of Spawn is the Dynasty[1] and OCEAN[6] systems, which avoid the communication overhead of auctions and use a pricing mechanism without any negotiation.

Other systems like Mats[9], Keren-Burak[14], Traveler[24], use specialized agents to gather information about the system state. Some systems use mobile agents to roam through the network to find free available resources, while others use agents to bid for resources. Both of these approaches share a common disadvantage in that they are time consuming. In the first approach, agents have to migrate from server to server until they find the needed resources – a difficult process in the absence of a pre-defined itinerary. Whereas in the second, an agent has to send a message (bid) to every available server and wait for a response (evaluation of the bid) from every server on the network. Although in this case the agent migration times are eliminated, the number of messages involved in every auction is likely to lead to a high network load.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC 2003, Melbourne, Florida, USA
© 2003 ACM 1-58113-624-2/03/03...$5.00
The Flash[7][16][Dr. Holger, personal email communication] system offers a framework for building distributed, load balancing applications in a heterogeneous environment, whereby a system agent maintains information about the whole system state and passes it to node agents on each server in the network. Node agents are required to maintain information about locally residing mobile agents. User agents (which are mobile) are responsible for the load balancing of the parallel application, and migrate through a cluster searching for free resources. Their migration decisions are based on internal states as well as internal and external events. The advantage of Flash is the reduction in the quantity of data transmitted over the network. This is achieved by prohibiting duplication of information and by eliminating unnecessary agent migrations i.e. any user agent can be aware of the system state information by just querying the system agent once. In contrast, the other approaches require every agent to query all the servers separately. Hence rapid acquisition of state information results in more reliable load balancing decisions.

1.2 Model-based Load Balancing
Some authors[2][12] have explored the model-based approach to load balancing, and use distributions of CPU load and expected process lifetime to decide if and when to migrate. Malone’s Enterprise[15] utilizes a market mechanism to support load balancing. Instead of using money, agents make bids giving the estimated time to complete a job. In contrast, the Challenger[3] system optimizes the load by introducing learning behavior in the bidding agents to deal with important parameters which have a major impact on system performance – such as message delay and error in estimating the job’s completion time. In Eager et al.[4] model-based load balancing processors do time-sharing i.e. they run more than one task at a time, in contrast with Enterprise and Challenger systems, which assume that processors are resources which can only be utilized by one task at a time. However, Eager’s algorithm suffers from the same shortcomings as Enterprise in that it cannot adapt, and is thus not robust under a wide range of operating conditions.

2. OVERVIEW OF THE SARA MAS
SARA is an active digital library of multi-spectral remote sensing images of the Earth from the SIR-C Shuttle mission, which provides Web-based online access to a library of data objects at Caltech, the San Diego Supercomputer Center, and the University of Lecce in Italy. The objective of the SARA project is to develop an infrastructure for a high-speed, high-volume, multi-protocol, distributed database, together with a means to attach distributed computing resources for data conversion, visualization and knowledge discovery[23]. A prototype MAS, which comprises both intelligent and mobile agents has been developed to manage and analyze distributed multi-agency remote sensing data[13]. The SARA[8] architecture is composed of a collection of information and Web servers, each of them having a group of agents. Information servers support Local Interface Agents (LIA), whereas Web servers support User Interface Agents (UIA). The information servers have the required computational resources and data repositories to support the SARA Digital Library (DL) – where the data repositories generally contain pre-processed images or geospatial data about a given region.

We separate mobile agents from stationary service agents in
SARA. Our approach is to localize the most complex functionality in non-mobile LIAs, which remain at one location, providing resources and facilities to lightweight mobile agents that require less processor time to be serialized, and are therefore quicker to transmit. LIAs are stationary agents that provide an extensible set of services. LIAs provide a level of abstraction between resource servers and the requesting mobile agents. The primary motivation for using mobile agents are: (1) The avoidance of large data transfers – of the order of Terabytes, consisting of sometimes proprietary data, (2) the ability to transfer user developed analysis algorithms, and (3) the ability to utilize specialized parallel libraries.

3. THE SARA LOAD BALANCE MECHANISM

The aim of load balancing in the SARA multi-agent system (MAS) is to evenly distribute agents among the servers, and to equitably serve the agents. The agents’ tasks are carried out simultaneously and there are no priorities between the agents, based on the time needed for their task to be accomplished. Agent task completion times therefore do not necessary imply a higher priority. The load balancing of the SARA MAS is based on a combination of the model-based and state-based approach, with the objective of minimizing the mean flow time, maximizing resource utilization and minimizing the mean response ratio\[18\].

Mean flow time is the average time from when a task is created to when it is completed. It is assumed that resources spend most of their time serving agents. Hence, resource utilization is the percentage of time a resource is used by agents to undertake their tasks. Mean response ratio is the average ratio of the actual time to complete a job divided by the time to run that task on an unloaded benchmarked server.

Decisions on load balancing are based upon a model which adapts due to the information gathered from the state-based approach. The model of load balancing in the SARA MAS is generic, and can be easily adapted for other MAS, especially those designed for Active Digital Libraries. The term \textit{active} implies that the Library provides computing services in addition to data-retrieval services, so that users can initiate computing jobs on remote supercomputers for processing, mining, and filtering of the data in the Library. The state-based aspect of load balancing is similar to the Flash\[7\][16] load balancing approach, as specialized agents are used which gather system state information and provide them to the mobile agents to assist with load balancing. The major difference between Flash and SARA for the state-based approach is that the load balancing decisions are made by the mobile agents. In Flash, mobile agents (\textit{user} as referred to in Flash) acquire system state information through specialized agents (system and node agents as referred to in Flash), and decide where they should migrate to, and at the same time balance the load of agents among the servers.

In the SARA system the load balancing decisions are supported through specialized agents referred to as \textit{management agents}. The objective of the management agents is to optimize the load on a given server i.e. tasks of all agents, whereas the mobile agents’ primary objective is to optimize execution of its own tasks. Although the mobile agents may be programmed to give priority to overall system optimization and not their own tasks, giving management agents the control over the load balance decisions leads to the following benefits:

i) \textbf{minimization of information transmitted over the network:} The management agents balance the load of mobile agents among the servers by defining their itinerary. Once an agent is created, it communicates with its local management agent, gives its requirements i.e. specifies its task, and waits for a response. The management agent in return, based on the agent’s requirements and the current system status, constructs the mobile agent’s itinerary and sends it back to that agent. Consequently, only two messages are exchanged between a mobile agent and a management agent: the agent’s requirements and the agent’s itinerary.

ii) \textbf{minimization of the mobile agent’s size:} Decisions on load balancing are based upon an algorithm (a model) that accepts as input an agent’s requirements and the system state information, and gives as output the server identity where the particular agent should migrate to. Management agents provide this functionality, and are stationary. Alternatively, the mobile agent would have to carry this decision support algorithm with it – which may require the migration of large agents across a network.

iii) \textbf{system optimization:} Management agents also maintain a record about mobile agents that are active on their host platform. This information includes the task of the agent, the resources that have been used, the time for completing the task and the site where the results of the task have been stored. This information is used to support load balancing, and for undertaking similarity analysis between agent requests. Hence, if an agent’s task (request) is identical to a task already performed, the task does not have to be repeated and previous results can be retrieved. If an agent’s task is similar but not exactly the same as an already accomplished task, the model determines if it is worthwhile for the agent to process the results of the existing task or to re-execute the task. Similarity analysis is undertaken based on the XML data model for encoding task properties (based on an overlap between points of a polygon – characterized by latitude/longitude coordinates).

Hence, management agents contribute to a mobile agent’s migration optimization by defining the itinerary for an agent according to its task and the current system state information. For instance, an agent with a task of acquiring a collection of images and filtering them on a compute server against a users custom analysis algorithm, will be guided by a management agent as to which server it should migrate to. How the agent migrates (i.e. the agent itself, the agent containing the custom algorithm etc) and when it should load any classes necessary for the accomplishment of its task, is its own choice. Even in the event of a server failure, the mobile agent is capable of moving autonomously to the next available server of its itinerary, communicating with the local stationary agents without being controlled by the management agents.

4. STATE-BASED LOAD BALANCING IN SARA

The SARA system is composed of a collection of information-servers and web-servers, each of them having a group of agents. A mobile agent (URA) is assigned to every client that needs to
access the system. As the number of clients increase so do the mobile agents. Essentially, the performance of the SARA system is based on the rapid and successful accomplishment of the mobile agents’ tasks. A management agent exists for every server. Every information-server has a LMA (Local Management Agent), where every Web-server has a UMA (Universal Management Agent). Although, the LMA and UMA management agents differ in their capabilities and the kind of information they process, their common objective is to optimize system performance.

Having a management agent on every Web-server from where URA agents are initially created, extends the possibility of optimizing the agents’ tasks and therefore the overall performance of the system. Once a URA migrates to an information-server, its itinerary can then be managed by the corresponding server’s LMA. Unnecessary agent moves e.g. migration to servers with unavailable resources, results in the delay of the agent’s task and contributes to the increase of network traffic. Hence, identification of an agent’s request and a comparison with a previous query is more preferable before the initialization of an agent’s itinerary. In the event of similar requests the UMA will decide if it is worthwhile for the agent to process the already stored results in order to accomplish its task.

Every LMA maintains information about its local server’s state and of other servers it interacts with. An LMA informs a visiting agent with respect to balancing load and the agent’s requirements i.e. to ensure that the next server the agent visits is of relevance with respect to data acquired at the current site. The use of multiple management agents over a centralized scheme is that the system does not have a single point of failure. If there is a failure in one of the management agents, the system can operate with all the remaining ones. Moreover, in the centralized scheme as the number of agents increase, the network load is increased dramatically due to the fact that all agents have to report to, and be managed by, a single management agent.

4.1 Information Maintained by Management Agents

The SARA architecture uses the global network map[5] for decentralized information distribution with a slight variation. The crucial part in this approach is the amount of information that has to be exchanged between management agents, since information must be replicated between them. The objective is to try and minimize information exchange without having a negative impact on the management agents’ capabilities.

Various management agents interact to ensure that their information is consistent. The greater the capabilities of a management agent, the greater the information needed to fulfill its duties. Consequently, the UMA needs more information than the LMA. Information must be replicated between all the management agents due to a limitation of global network map approach. Furthermore, details of the mobile agent needed by the UMA (but not by the LMA) has to be exchanged between all of the management agents. The omission of this information not needed by the LMAs, on the LMA-LMA and LMA-UMA interaction would result in minimization of information transfers and therefore reduction of network overload.

<table>
<thead>
<tr>
<th>Table 1. LMA’s information</th>
<th>acquired by</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMA’s information</td>
<td></td>
</tr>
<tr>
<td>local LAA</td>
<td></td>
</tr>
<tr>
<td>LMA itself</td>
<td></td>
</tr>
<tr>
<td>LMAAs</td>
<td></td>
</tr>
<tr>
<td>sender agent</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2. UMA’s information</th>
<th>acquired by</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMA’s information</td>
<td></td>
</tr>
<tr>
<td>local UAA (upon URA’s creation)</td>
<td></td>
</tr>
<tr>
<td>local UAA (before URA’s death)</td>
<td></td>
</tr>
<tr>
<td>UMAAs</td>
<td></td>
</tr>
</tbody>
</table>

Our variation of the global network map lies in the information distribution between agents. Records held by each of the management agents has a tree structure and encoded in XML – as in tables 1 and 2. The second column on the right side of each of the tables identifies the source from which information is acquired. This corresponds to two global network maps; the LMA’s global network map nested into the UMA’s global network map (represented by the gray color in the tables). The
limitation lies in the information replicated between similar management agents i.e. LMA-LMA, UMA-UMA, (colored in grey, see table 2) exchanged between LMA-UMA interaction.

The management agents’ (LMAs) information is divided into two sections: the first contains information for the local server represented by the “local” label, and the second contains information for the rest of the servers represented by the “remote” label. Therefore, every LMA maintains system state composed of local and remote information. Similarly, the first section of the UMA’s information contains personal details of the local URA agents i.e those that have been launched from the server where the particular UMA resides. The second section contains two kinds of information. The first part is limited (i.e only those details needed for identification of request similarities) of the URA agents launched from other Web-servers. The second part has details of every LMA’s i.e. the current system state information. During the management agents’ interaction only the “local” information is exchanged.

The messages exchanged between the agents are time-stamped. Therefore, the bandwidth between two servers is estimated by dividing the transmission time of a message (from the sender server to the receiver) by the amount of data transmitted. The information regarding the server’s bandwidth (row 4 in table 1) is acquired at the initialization of the system where all the management agents from each server exchange their information between themselves with time-stamped messages. The bandwidth between two servers is updated only when there is a significant deviation between the last known recorded bandwidth of those servers and the bandwidth derived from a (time-stamped) message exchanged between a sender and receiver agent.

Hence using two different network maps reduces message traffic and results in a decrease of network load. When there is an update to the URA’s personal details, interaction is only between UMA themselves and does not involve the LMA. Furthermore, based on the limitation of this approach, the amount of information exchanged is also reduced, as on the LMA-UMA interaction only the LMA’s information (colored in grey, see table 2) is exchanged.

### 4.2 Communication Between Management Agents

The management agents exchange information via direct or multicast messages depending on the number of participants that are involved in the message exchange. Every management agent has its own Space[21] (Voyager’s[19] optimized technique for multicast message exchange), in which its contents are the union of the rest of the LMA and UMA management agents. Although, every UMA has an extra Space consisting only of UMAs, as there are occasions where information exchange occurs only between UMAs. At initialization, the management agents exchange all of their information between themselves; subsequently only the updated information is exchanged. Interaction and exchanges between management agents are based on system events summarized in table 3. Table 3 also identifies the interactions between the management agents and the rest of the agents that update their information.

### 5. MODEL-BASED LOAD BALANCING IN SARA

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 where the particular agent should migrate to. Since load balancing is controlled by the management agents, this algorithm (the model) is maintained by the UMA and LMA management agents. The model is a function of the: (1) agents’ tasks, (2) servers’ utilization (performance load), (3) availability of resources at the server, and (4) network

---

**Table 3. Management agents’ interaction**

<table>
<thead>
<tr>
<th>Event</th>
<th>Interaction</th>
<th>Information exchange</th>
<th>Type of mes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>on the initialization of the system</td>
<td>LMA-LMAS/UMAs</td>
<td>contents in row 1.2 of table 1</td>
<td>multicast</td>
</tr>
<tr>
<td>upon URA’s creation</td>
<td>UAA-local UMA</td>
<td>contents in row 1 of table 2, information in bold of table 2</td>
<td>direct, multicast</td>
</tr>
<tr>
<td>before URA’s death</td>
<td>UAA-local UMA</td>
<td>contents in row 2 of table 2, information in bold, in row 2 of table 2</td>
<td>direct, multicast</td>
</tr>
<tr>
<td>URA’s migration failure</td>
<td>URA-local LMA/UMA</td>
<td>Voyager server is down (row 1, table 1)</td>
<td>direct, multicast</td>
</tr>
<tr>
<td></td>
<td>LMA/UMA-LMA/UMAs</td>
<td>database is unavailable (row 1, table 1)</td>
<td>direct, multicast</td>
</tr>
<tr>
<td></td>
<td>LMA-LMAS/UMAs</td>
<td>the time the server will become available (row 1, table 1)</td>
<td>multicast</td>
</tr>
<tr>
<td></td>
<td>UMA-UMA</td>
<td>selected information of row 1.2 of table 2 based on the recipient UMA needs</td>
<td>direct</td>
</tr>
<tr>
<td></td>
<td>LAA-local LMA</td>
<td>contents in row 1 of table 1</td>
<td>direct, multicast</td>
</tr>
<tr>
<td></td>
<td>LMA-LMAS/UMAs</td>
<td>contents in row 1.2 of table 1</td>
<td>direct, multicast</td>
</tr>
<tr>
<td></td>
<td>UMA-UMAs</td>
<td>contents in row 3 of table 2</td>
<td>multicast</td>
</tr>
</tbody>
</table>
Cases 1 to 5 (figure 2) occur when a task is similar to one performed by another agent in the past, and where results have been stored on a server in a known location. Cases 5 to 7 occur when a task has not been performed previously. More complex tasks can be supported by combining a sequence of simpler tasks defined by the tree structure. Below it is explained in detail, how the optimum itinerary of an agent that its task falls to Case 2 is constructed i.e. to which server the agent should migrate in order to accomplish its task, conducting at the same time in load balancing.

In Case 2, the agent’s task is exactly the same to a task performed in the past. Furthermore, the results acquired by an agent might need filtering (as an extension of the agent’s task) by a filter that exists on a compute server or by a custom analysis filter provided by a user. In order to achieve effective load balancing, the model should also account for possible combinations of an agent’s tasks.

Cases 1 to 5 (figure 2) occur when a task is similar to one performed by another agent in the past, and where results have been stored on a server in a known location. Cases 5 to 7 occur when a task has not been performed previously. More complex tasks can be supported by combining a sequence of simpler tasks defined by the tree structure. Below it is explained in detail, how the optimum itinerary of an agent that its task falls to Case 2 is constructed i.e. to which server the agent should migrate in order to accomplish its task, conducting at the same time in load balancing.

In the event where the agent’s task is the same with a task of another agent which is still in progress, the former agent is deactivated until the later agent finish its task. The UMA management agents maintain information about the progress of every agent task. Therefore, in such event the management agent instructs the former agent to become persistent and notifies the later agent that another one has the same task with it. When the later agent finishes its task, it sends a message to the former agent with details of where the results of the task can be retrieved from.

Although, investigation is needed in the occasion where the later agent has already finished its task but the server $S_1$ is now temporarily unavailable. Therefore, the itinerary of the agent is influenced by the status of the $S_1$ server. Figure 3 presents the different cases that need to be examined, according to the status of $S_1$ server.

\[
\begin{align*}
\text{Case 2i} & : \quad \text{Server is available} \\
\text{Case 2ii} & : \quad \text{Server will become available at } T_s \\
\text{Case 2iii} & : \quad \text{Server unreachable for unknown time}
\end{align*}
\]

Figure 3. The sub-cases of the case 2 agent’s task

\textit{case 2i: } In this case, the server on which the results have been stored is available (online). Hence, the agent should retrieve the results directly from their physical location, as described above.

\textit{case 2ii: } In this case the server is currently unavailable but will become available at time $T_s$. Consequently, the point is to check if it is worthwhile for the agent to be deactivated on the server where it currently resides. Alternatively, it may move to $S_1$ that will hold the result. In order to decide if the agent should be deactivated or not, we should compare $T_s$ with the minimum time needed for the agent’s task to be accomplished on another server, denoted by $T_{min}$. We can calculate $T_{min}$ as follows:

Let’s denote:

\[T_n = \text{the time taken to complete a task, on the currently unavailable } S_1 \text{ server.} \]

\[U_{ps} = \text{server } S_1 \text{ ’s utilization, when the task has been accomplished.} \]

\[U_s = \text{a server’s utilization.} \]

We know that the task which the agent needs to complete needed $T_n$ time by another agent on the currently unavailable server $S_1$ server with known $U_{ps}$. Since a machine of speed 2 should originate twice as many jobs as a machine of speed 1[3], we can estimate how long it will take for an agent to accomplish the same task on a different server on a given $U_s$. Therefore for each of the available servers with the appropriate resources we calculate:

\[
\begin{align*}
\text{Case 2iv: } & \quad \text{Server is available} \\
\text{Case 2v: } & \quad \text{Server will become available at } T_s \\
\text{Case 2vi: } & \quad \text{Server unreachable for unknown time}
\end{align*}
\]
\[ x = T_{ts} \times \frac{U}{U_{ps}} \]

and for the rest of the servers that are unavailable but will become available at \( T_s \) time and have the appropriate resources we add \( T_s \) time for the corresponding server. Note that when the currently unavailable servers come online, their utilisation will be zero, as they will have no agent tasks to perform. Furthermore, for each of the servers evaluated in Formula 1 we have to add the time the agent needs to migrate itself from its current position to that server. For the unavailable servers we assume that the bandwidth between the server where the URA agent currently resides and the unavailable server is the same as the last known bandwidth recorded by the management agent.

The \( x \) with the minimum value gives the \( T_{\min} \), where the server with the corresponding \( T_{\min} \) is denoted by \( S_2 \). If \( T_{\min} \) is greater or equal to \( T_s \), then the agent should be deactivated and wait for the unavailable server \( S_1 \) to come online in order to acquire the results for its task directly from the physical location where they have been stored. As server \( S_1 \) will become online (at \( T_s \) time) before the agent could finish its task (\( T_{\min} \) time) on \( S_2 \) server. Therefore, the agent by being deactivated and not migrating to \( S_2 \) server, firstly will acquire the results faster and secondly it will not influence the load balance or the utilisation of \( S_2 \) server i.e. by using \( S_2 \) server’s resources to accomplish its task. Even if, \( T_{\min} \) is equal to \( T_s \), the agent will acquire the results for its task at the same time, regardless to which server it will migrate to (\( S_1 \) or \( S_2 \)). If \( T_{\min} \) is less than \( T_s \) the agent should be deactivated unless the difference is larger than a threshold (assuming is greater than \( T_{\min} + T_{\min}/2 \)) so that \( T_{\min} \) is much less than \( T_s \). Although, the agent by migrating to \( S_2 \) server will influence the server’s load and utilisation, it will accomplish its task in \( T_{\min} \) which in this case is much less than \( T_s \) (the time needed for \( S_1 \) server to become online). This ensures that agents are also equally served.

\textit{case 2ii:} In this case, the server is currently unavailable and it is not known for how long it will remain offline. Therefore, we list the servers with the appropriate resources which are either online or currently unavailable but which will become online at \( T_s \) time. For each of the listed servers we calculate the \( x \) value, as in case 2ii. The \( x \) with the minimum value denotes the corresponding server to which the agent should migrate to.

5.1 Assumptions of the Model

The model is based on the following assumptions: The utilisation of a server is measured based on Malone’s\cite{15} definition, which is also used by the Challenger\cite{3} and Flash\cite{11} projects:

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

where:
\( \alpha \) = the number of agents on that server
\( \mu \) = the average task time of the \( a \) agents
\( L \) = the processing power of the server

The above formula concerns only the active URA agents that exist on a server. Thereby, on the initialization of a server its utilisation is assumed to be almost zero because it does not have any URA agents. The persistent agents are not taken into account since they do not consume any vital resources. In the Voyager platform the persistent agents are considered as object of a database\cite{20}, therefore the only processing power they require is during their transaction from active to persistent (deactivated state) and vice-versa. On the calculation of a server’s utilisation that does not have any agents we assume that there exists one agent with an average task time of 0.1 seconds. In the event where there are two servers with no agents but different processing power it does not mean that if an agent migrates to either of those two servers it will require the same execution time. Therefore the utilisation of a server when it has no agents should not be equal to zero. By setting the numerator of the formula to a non-zero value the utilisation of a server is influenced by its actual processing power i.e. \( L \), the dominator of the formula’s fraction. In the case where there are two servers with the same utilisation and the appropriate resources for serving an agent’s task the management agent will chose the one with the less available resources i.e. the server that has only those resources needed for the completion of the agent’s task. Even if both of the servers have the same characteristics then the selection is made in random. If there no sever available with the appropriate resources to process an agent’s task then the agent becomes persistent and is informed by its local management agent to which server it should migrate as soon as a server that can handle the agent’s task becomes available. An agent may remain persistent for a specific period of time – subsequently a failure message is generated if it is not activated.

Another assumption concerns the filtering of the agents’ results. The filtering against a custom analysis algorithm (filter) is performed only on a compute server. The distance between the server where the results of the agents’ tasks are stored and the compute server is assumed to be null i.e. the compute server can directly (locally) filter any results stored by an agent on a server of the same information server. Finally, the information regarding the URA agents maintained by the UMA management agents are kept until the user deletes its files i.e. the results gathered from his/her representative URA agent or during the LAA agent’s file-space maintenance check.

6. CONCLUSION

Load balancing in SARA is a combination of the state-based and model-based approaches. Decisions on load balancing are based on the information gathered from the state-based approach and on probabilistic estimation used in the model of the model-based approach. There will undoubtedly be errors in the estimation of the model but due to the information on the progress of the URA agents and the observation of the whole system provided by the UMA management agents, it is possible to optimize the intelligence of the management agents for improving their accuracy on load balancing decisions. For instance, based on the information gathered by the management agents we can deduce if an agent by migrating to another compute server for accomplishing its task faster influences negatively or positively the tasks of the other agents. The use of similarity analysis between agent queries can also support load balancing by preventing tasks to be re-run.
7. REFERENCES


