Journal of Theoretical and Applied Information Technology
31st October 2018. Vol.96. No 20
© 2005 – ongoing JATIT & LLS

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A SCALABLE SERVICE ALLOCATION NEGOTIATION FOR CLOUD COMPUTING

[1]LESZEK SLIWKO

[1]Faculty of Science and Technology, University of Westminster

E-mail: lsliwko@gmail.com

ABSTRACT

This paper presents a detailed design of a decentralised agent-based scheduler, which can be used to manage workloads within the computing cells of a Cloud system. This scheme in based on the concept of service allocation negotiation, whereby all system nodes communicate between themselves and scheduling logic is decentralised. The architecture presented has been implemented, with multiple simulations run using real-word workload traces from the Google Cluster Data project. The results were then compared to the scheduling patterns of Google's Borg system.

Keywords: *Distributed Scheduling; Agents; Load Balancing; MASB*

1. INTRODUCTION

Cloud computing has become a very widespread phenomenon in daily lives, especially if we consider how accepted smart phones are. A typical phone application is heavily depended on infrastructure and remote processing capabilities Clouds deliver. Also, in desktop computer's programs area, a significant number of applications have been moved into Software-as-a-Service model, where desktop serves only as a client, but actual processing in done on a remote Cloud, e.g.: business subscriptions for Microsoft Office 365 offer access to cloud-hosted versions of Office programs. Therefore, the current world is more and more reliant on Cloud computing.

A variety of vendors, services and business models have created an extremely complex environment. The latest developments in Cloud technologies gravitate towards federated, inter-Cloud cooperative models and, therefore, already very complex solutions are destined to become even more complex. It's is also a very competitive market. A recent rise in Big Data systems fuelled a growth in demand of cheap computing power; however, a number of vendors joined and market offering computing services has greatly expanded Prices have also been driven down and, as of 24 July 2018, the cost of renting a general-use instance of 16-core machine with 64GB memory was 80 cents per hour (data from aws.amazon.com/ec2/pricing website).

The main difference between a Cluster and a Cloud is business model and access. Clusters were traditionally available only to a very limited number of institutions and corporations, who pooled their resources in order to get advantage of scale of computing. Public generally did not have access to computing capabilities of Clusters.

Because access to Clusters was very restricted and incoming workload was planned well in advance, it was practical to centralise task scheduling and system management functions. Therefore, a centralised architecture was generally used a base for Cluster management software, such as SLURM [61], Univa Grid Engine [16], Google's Borg [44], etc.

Cluster schedulers evolved from a sequential processing of allocation decisions in a loop (e.g. SLURM) towards more parallel solutions [44]:

(i) Statically partitioned scheduling in which the exclusive sets of machines in a Cluster are dedicated to certain types of workload and the resulting specialised partitions are managed separately. The example of such design is Microsoft's Quincy [39]. The main criticism of the static partitioning is inflexibility – a homologous workload might result in a part of scheduler being relatively idle, while other nodes are very active.

(ii) Two-level architecture in which a Cluster is partitioned dynamically by a central coordinator. The actual task allocations take place at the second level of architecture in one of the specialised schedulers. The first two-level scheduler was Mesos [20] in which resources are distributed to the frameworks in the form of 'offers' made by Mesos Master. Scheduling frameworks have autonomy in deciding which resources to accept and which tasks to run on them. The mechanism used causes

resources to remain locked while the resources offer is being examined by a specialised scheduler. This means the benefits from parallelisation are limited due to pessimistic locking. In addition, the schedulers do not coordinate between each other and must rely on a centralised coordinator to make them offers, which further restricts their visibility of the available resources in the Cluster.

(iii) Shared state is the most recent design, used in Google's Borg [44] and Microsoft's Apollo [7]. The concept behind this architecture is to deploy several schedulers working in parallel. The scheduler instances are using a shared object with a state of available resources; however, the resource offers are not locked during scheduling decisions (optimistic concurrency control). In the case of a conflict, when two or more schedulers allocated jobs to the same resources, all involved jobs are returned to the jobs queue and scheduling is re-tried.

This research focuses on the next type of scheduler architecture, which expands on the shared state scheduler concept where the scheduling logic is processed in parallel but still within a specific entity. The key novelty of the presented design is the distribution of the scheduling logic's processing to Cluster nodes themselves – this approach eliminates the requirement that all allocation decisions have to be synchronised and aggregated into a single state. Such a solution should scale beyond the limits of the currently deployed Clusters orchestration software while preserving or bettering the workload throughput and quality of centralised scheduler's allocations. The key objective of this research was to experimentally evaluate performance advances emerging out of the designed solution.

One of the most developed and published Cluster managers is Google's Borg system which represents the shared state scheduler architecture. When allocating a task, Borg's scheduler is scoring a set of available nodes (best-fit algorithm) and selects the most feasible machine for this task. The central module of Borg architecture is BorgMaster, which maintains an in-memory copy of most of the state of the cell. Each machine in a cell is running BorgLet, an agent process responsible for starting and stopping tasks and also restarting them if they fail. A single BorgMaster controller is able to manage a cell of more than 12k machines (highest value is not specified). Google's engineers achieved this impressive result by a number of optimisations and, so far, they've managed to eliminate or work around virtually every limitation they have approached [51]. Nevertheless, a centralised architecture is a limitation within a current design of a Cloud.

One might also ask a fundamental question - do Cloud systems really require more inter-connected nodes in one cell? Computing power of 12k machines working together is already a very considerable force and, barring few exceptions, it's highly unlikely that an application would require such a processing power. However, in recent years, software development is trending towards Big Data systems. Big Data systems are characterised by a high degree of parallelism. A typical Big Data system design is based on a distributed file system, where nodes have dual function of storing data as well as processing it. One process of such system might need to crunch thousands of TBs of data split across thousands of nodes. Even with ideal allocation of Big Data tasks, where every task is processing data only available locally (i.e. locality optimisation), a single node would still need to process GBs of data locally. Therefore, the answer to the above question is yes – it's highly likely that we will require larger computing cells in the near future.

Therefore, if centralised architecture has its ultimate scalability limits, we shall consider alternative approaches. In this paper we present a working prototype of decentralised Cloud manager – Multi-Agent System Balancer (MASB), which relies on a network of software agents to organically distribute and manage good system load. MASB prototype has been built on top of Accurate Google Cloud Simulator (AGOCS) framework [48] and such all research and development has been continuously tested on a real-world workload traces from the Google Cluster Data (GCD) project [19].

The main novel aspects of this approach were to co-operatively schedule the incoming tasks by a network of software agents, which allows running programs to be offloaded to alternative system nodes on the fly, in addition to designing algorithms capable of proactively managing a workload in such a dynamic environment. Thus, this research breaks with the concept that the execution of a task in a cluster is immovable or unstoppable, and instead examines the available technology to implement such a strategy.

Furthermore, moving away from the concept of the centralised load balancing and offloading the actual scheduling logic to the nodes themselves resulted in more time available for the execution of allocation routines. As such, more sophisticated algorithms could be deployed, such as metaheuristic methods. Since none of the commercially available cluster schedulers realise such features, the objective of this research was to implement a working prototype for the Cloud load balancer, and to

evaluate their performance advances emerging out of the designed solution.

The remaining of this paper is organised as follows. Section 2 provides introduction to agent systems and a brief of research history how they have been used to schedule tasks. Section 3 describes MASB's design principles, defines scope of project and lists all main used technologies. Section 4 has details about MASB architecture and its objects. Section 5 describes in detail the allocation negotiation protocol that is used between system components to allocate new task or re-allocate existing task on a node. Section 6 present allocation score functions, which are modelling distribution of tasks and their allocations across nodes through tasks' lifecycles. Section 7 specifies experiments, which were performed as a part of his research and provides a discussion on achieved results. Section 8 describes competitive solutions. In section 9, important optimisations and lessons learnt are presented.

2. LOAD BALANCING WITH AGENTS

Agent technologies can be dated back to 1992 [42], at which point it was predicted that intelligent agent would become the next mainstream computing paradigm. Agents were described as the most important step in software engineering, representing a revolution in software [18]. Since its inception, the field of multi-agent systems has experienced an impressive evolution, and today it is an established and vibrant field in computer studies. The software agents research field spans many disciplines, including mathematics, logic, game theory, cognitive psychology, sociology, organisational science, economics, philosophy, and so on [54]. Agents are considered to be a viable solution for large-scale systems, for example through spam-filtering and traffic light control [9], or by managing an electricity gird [8].

It is difficult to argue for any precise definition of an agent, with the research literature seeming to suggest that there are four key properties of an Agent [13][15][56], namely:

- Autonomy when allowing agents to operate without direct human intervention;
- Social ability when agents communicate and interact with other agents;
- Reactivity when agents actively perceive their environment (physical or digital) and act on its changes;
- Proactiveness when agents not only dynamically respond to changes in environments but are also able to take initiative and exhibit goal-oriented behaviour as well as real-time communications.

A software agent it is generally defined as being of acting independently of its user in order to accomplish tasks on behalf of its user [36]. An agent can be described as a being which is supposed to act intelligently according to environmental changes and the user's input [17].

Software agents are found across many computer science disciplines, including AI, decentralised systems, self-organising systems, load balancing and expert systems [18][30]. Previous research has also shown that by deploying agents it is possible to achieve good global system performance [34] and attain dynamic adaptation capability [23].

Agents were also found to be useful for the performance monitoring of distributed systems [10]. Several additional benefits may also be achieved, including more cost-effective resource planning [11], a reduction of network traffic [32], the autonomous activities of the agents [17], and decentralised network management [59]. Multi-agent systems were also successfully used for forecasting demand and then adapting the charging schedule for electric cars [58], and also to effectively coordinate emergency services during crisis [37]. [40] presents an agent-based framework to model procurement operations in India. The most state-of-art research generally focuses on negotiation protocols and communications [28][31][53][57].

Agent-based systems generally rely on decentralised architecture [22][31][46][53], considering it to be more reliable. However, those schemas require complex communication algorithms, with negotiation protocols often being required for distributed architecture to attain a good level of performance [6][57][59].

The idea of job scheduling with agents is not new; a single-machine multi-agent scheduling problem was introduced in 2003 [3][4]. Since this time, the problem has been extended and exists in several variations, such as deteriorating jobs [26], the introduction of weighted importance [35], scheduling with partial information [27], global objective functions [50], and adding jobs' release times and deadlines [60]. A suitable taxonomy of multi-agent scheduling problems in presented in [38].

The research on workload sharing via agents has a long history, with the papers below in particular having influenced the design of the MASB:

(i) [43] presents a study concerning a multi-agent system in which all decision making is performed by a learning AI. The likeness of selection of a particular node for the processing of a given task depends on the past capacity of this node. The Agent's AI uses only locally-accessible knowledge, meaning that it does not rely on information shared by other agents.

(ii) [14] introduces Challenger, a multi-agent system, in which agents communicate with each other to share their available resources in an attempt to utilise them more fully. In Challenger, agents act as buyers and sellers in a resources marketplace, always trying to maximise their own utility. MASB follows a similar pattern, where nodes try to maximise their utilisation (via score system).

(iii) [6] shows that cooperative negotiation between agents representing base stations in a mobile cellular network can lead to a near global optimal coverage agreement within the context of the whole cellular network. Instead of using a negotiation model of alternating offers, several possible local hypotheses are created, based on which parallel negotiations are initiated. The system commits to the best agreement found within a defined timeline. The cooperative model in which agents negotiate between themselves is the base of the distributed scheduling presented in this research.

(iv) [23] proposes a load-balancing scheme in which a mobile agent pre-reserves resources on a target machine prior to the occurrence of the actual migration. The system also prevents excessive centralisation through the implementation of a mechanism whereby when the workload processed on a particular machine exceeds a certain threshold, this machine will attempt to offload its agents to neighbouring machines.

(v) [12] describes a solution in which agents representing a local grid resource uses past application performance data and iterative heuristic algorithms to predict the application's resource usage. In order to achieve a globally-balanced workload, agents cooperate with each other using a Point-to-Point (P2P) service advertisement and discovery mechanism. Agents are organised into a hierarchy consisting of agents, coordinators and brokers, who are at the top of the entire agent hierarchy. The authors conclude that for local grid load balancing, the iterative metaheuristic algorithm is more efficient than simple algorithms such as FCFS.

(vi) [21] details a solution built on top of the ant colony algorithm, a solution which takes its inspiration from the metaphor of real ants searching for food. 'Ants' are software objects that can move between nodes managed by agents. A move between nodes which is managed by the same agent is less costly. Ants explore paths between nodes, marking them with different pheromone strength. Whenever an Ant visits a node, the agent managing it saves the recorded tour and updates its own database. Ants who subsequently visit this node read its current knowledge, meaning they have the potential to exchange information in this environment, which adds to the predictability of the whole solution.

3. MASB DESIGN PRINCIPLES

The MASB project has been developed over several years, during which time it has undergone many changes in terms of both the technology used and the design of the architecture. This has included, for example, migration from Java to Scala, the change from thread pools to an Akka Actors/Streams framework, and the introduction and use of concurrency packages and non-locking object structures. However, the main design principles have not been altered and are presented below:

- To provide a stable and robust (i.e. no single point of failure) load balancer and scheduler for a Cloud-class system;
- To efficiently reduce the cost of scaling a Cloud-class system so that it can perform in an acceptable manner on smaller clusters (where there are tens of nodes) as well on huge installations (where there are thousands of nodes);
- To provide an easy way of tuning the behaviours of a load balancer where the distribution of tasks across system nodes can be controlled.

Many other Cluster managing systems, such as Google's Borg [51], Microsoft's Apollo [7] and Alibaba's Fuxi [63], were built around the concept of the immovability and unstoppability of a task's execution. This means that once a task is started it cannot be re-allocated: it can only be stopped/killed and restarted on an alternative node.

This design is particularly well suited when there is a high task churn, as observed in Apollo or Fuxi where tasks are generally short-lived, meaning that the system's scheduling decisions do not have a lasting impact. However, in order to support a mixed workload which features both short-lived batch jobs and long-running services, alternative solutions needed to be developed. One such solution is the resource recycling routines present in Borg wherein resources allocated to production tasks but not

currently employed are used to run non-production applications [51].

MASB takes advantage of virtualisation technology features, namely Virtual Machine Live Migration (VM-LM), to dynamically re-allocate overloading tasks. VM-LM allows programs which are running to be moved to an alternative machine without stopping their execution. As a result, a new type of scheduling strategy can be created which allows for the continuous re-balancing of the cluster's load. This feature is especially useful for long-term services which initially might not be fitted to the most suitable node, or where their required resources or constraints change. Nevertheless, this design creates a very dynamic environment in which it is insufficient to schedule a task only once. Instead, a running task has to be continuously monitored and re-allocated if the task's current node cannot support its execution any longer.

The design of MASB relies on a number of existing tools and frameworks. The main technologies used are listed below:

(i) Decentralised software agents – a network of independent AI entities that can negotiate between each other and allocate Cloud workload between them. In MASB, specialised agents control nodes and manage the system workload. Due to the decentralised nature of MASB, there is no complete up-to-date system state. Instead, yet another type of agent is responsible for caching the nodes' statistics and providing an interface whereby a set of candidate nodes which a particular task can be migrated to can be requested.

(ii) Metaheuristic selection algorithms – while the majority of the processing of load balancing logic is done via negotiation between agents, a few system processes are handled locally. One such example is that when an agent discovers its node is overloaded, it will select a subset of its tasks which it will attempt to migrate out. This selection is performed by Tabu Search (TS) algorithm.

(iii) VM-LM which allows the transfer of a running application within the Virtual Machine (VM) instance to an alternative node without stopping its execution. The vendors' strategy is to implement mixed production and low-priority jobs on a single machine. While production jobs are idler, low-priority jobs consume the nodes' resources. However, when production job resources need to be increased, the low-priority jobs are killed. The non-production jobs in Google Cluster [51] and the spot-instances in Amazon EC2 [52] use such an approach. MASB takes advantage of VM-LM to offload tasks without stopping their execution, collecting information about tasks in order to estimate the VM-LM cost of such a task.

(iv) Functional programming language Scala and accompanying libraries – due to the decentralised design and loose coupling between the system's components, the implementation language is of secondary importance. However, load balancing algorithms require a significant amount of tuning, especially if the Cloud is designed to have a high utilisation of available resources. This would mean that resource waste is low, and therefore the cost-per-job execution is also low. Due to the complexity of inner-system relations and dependencies, a high-fidelity simulation environment is necessary to evaluate the expected performance of a given configuration and implemented changes before is deployed to a production system, e.g. the FauxMaster simulator used by Google Engineers [51]. In this implementation, Akka Actors framework was selected as the core parallelisation technology.

4. MASB ARCHITECTURE

The experiments in [49] that used a centralised load balancer based on metaheuristic algorithms demonstrated that, due to the high overheads of these algorithms, a scheduling strategy implemented on a single machine is highly unlikely to efficiently manage a large number of tasks. Therefore, MASB has been built around the concept of a decentralised load balancing architecture, an architecture which could scale well beyond the limits of a centralised scheduler.

Figure 1 visualises the communications' flow within MASB system:

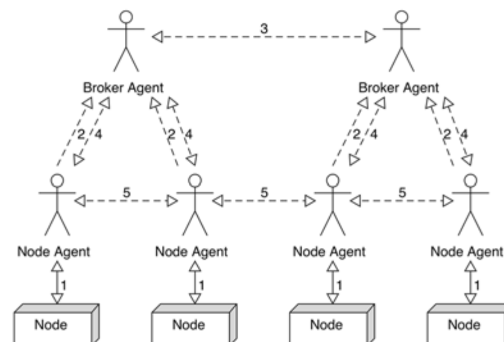

Figure 1. MASB communications' flow

MASB relies on a network of software agents to organically distribute and manage the sizeable system load. All communication between the agents is performed via a specialised stateless P2P protocol

which promotes loose coupling. Two types of agents are deployed: Node Agent (NA) and Broker Agent (BA). NAs are supervising system nodes, are responsible for keeping those nodes stable. NAs actively monitor the used resources on their nodes (1) and periodically forward this information to the subnetwork of BAs (2). BAs continuously exchange nodes' load information between themselves (3) and, therefore, effectively cache the state of the computing cell.

NA contains an AI module which is based on a metaheuristic algorithm TS. It manages a workload on a node. When an NA detects that its node is overloaded, it will attempt to find an alternative node for overloading tasks with the help of Service Allocation Negotiation (SAN) protocol (the details can be found in section 5). The first step of SAN communication is to retrieve alternative nodes from BA (4). BAs provide a query-mechanism for NAs, which returns a set of candidate nodes for the migrations of tasks. However, because the information found in BAs is assumed to be outdated, once the NA completes this step, it communicates directly with their NAs so as to re-allocate this task (5).

The following two subsections describe the types of agents noted and detail their responsibilities. The annotated arrows 2 to 5 in Figure 1 correspond to inter-agent communications – messages that are exchanged within the system are detailed in subsection 4.3.

4.1 Node Agent

Every node in the system has a dedicated instance of NA. NA continuously monitors the levels of defined resources and periodically reports the state of its node and levels of utilised resources to BAs. Should any of the monitored resources be over-allocated, NA will initialise SAN process. In addition, NA performs the following functions:

(i) Accept/deny task migration requests – NA listens to task migration requests, and accepts or denies them. This routine is simple, with NA projecting its resource availability with that task as follows: projected allocation of resources = current allocation of resources (existing tasks which also includes tasks being migrated out from this node) + all tasks being migrated to this node + requested task (from request). If the projected resources do not overflow the node, the task is accepted and the migration process is initiated. The source node does not relinquish ownership of the task while it is being re-allocated, meaning that source node is regarded as a primary supplier of the service until the migration process successfully completes. It should be noted that during task migration, its required resources are allocated twice, to both the source node and the target node.

(ii) Task migration – after accepting the task migration request, NA immediately starts listening for incoming VM-LM. In order to perform task migration, NA must have access to the administrative functions of VM and be able to initiate VM-LM to another node. This functionality can be either implemented by the calls of the VM manager API or by executing the command line command. This process may vary considerably per VM vendor.

4.2 Broker Agent

BA is responsible for storing and maintaining information about nodes' online status and their available resources. BA is a separate process which can coexist with NA on the same node since its operations are not computing-intensive. BA has two main purposes in the system. These are outlined below:

(i) Nodes resources utilisation database – NA periodically reports to its BA about the state of its node and available resources. BA stores all this data and can query them on demand. Every node entry is additionally stored with its timestamp, showing how long ago the data were updated. It has additional protection against the node silently going offline, for example through hardware malfunction or the network becoming unreachable, in that if this entry is not updated for five minutes, the node is assumed to be offline and entry is removed. This means that it will not be returned as the candidate node.

(ii) Evaluating candidate nodes for a task migration – BA listens for requests and computes a list of candidate nodes for a task migration. In order to create a list of candidate nodes, BA retrieves nodal data from the local cache and then scores them using Allocation Scoring Function. BA scores the future state of the system as if task migration were being carried out. After scoring all the cached nodes, BA selects a configured number of candidate nodes with the highest score and sends them back to the asking node. In this research this number was set to fifteen candidate nodes, wherein higher numbers failed to yield superior results.

4.3 Message Types

In order to avoid costly broadcasts, since broadcast packages need to be rerouted through a whole network infrastructure consuming the available bandwidth, both NA and BA always communicate P2P. Agent-to-agent interactions

follow the 'request-response' pattern, in which each request object has one or more matching response objects. The message objects carry additional metadata such as fitness value (see formula subsection 5.1), forced migration flag, and detailed node and task information. Section 5 explains the process in which messages are exchanged, while the subsections 5.1 to 5.4 show detailed samples of such objects. In the system there are several types of requests and responses between agents outlined in Table 1 below:

Request Type	Description
GetCandidateNodesRequest	Requests a number of candidate nodes for the migration of a specified tasks set. Send from NA to BA.
GetCandidateNodesResponse	Reply with a set of candidate nodes for task migration, together with their resource statistics.
TaskMigrationRequest	Request from source NA to candidate NA as to whether task migration is accepted.
TaskMigrationAcceptanceResponse	Replay from target candidate NA that task migration will be accepted. Note: No resource allocation takes place after this request.
TaskMigrationRejectionResponse	Replay from target node's NA that task migration will not be accepted.
TaskMigrationProcessRequest	Request to selected target node's NA to start task migration. Note: this request has an optional forced flag, requesting the target NA to skip the currently available resources check. The total node's resources check and constraints check will be still performed.
TaskMigrationProcessConfirmationResponse	Confirmation from the target node's NA that the task migration process can start. Note: Resources are allocated for the migrated task and the live migration process starts.
TaskMigrationProcessErrorResponse	Denial of task migration process. This reply is generated if the NA can no longer accommodate the migrated task.

Table 1. Message types

5. SERVICE ALLOCATION NEGOTIATION PROTOCOL

When NA detects its node is overloaded, it will select a task (or a set of tasks) and attempt to migrate them to an alternative node or nodes. Since SAN is asynchronous, this means a single NA can run several SAN processes in parallel. In the current implementation, NA selects a number of tasks in the first step – Select Candidate Services (SCS) – and processes their allocation in parallel.

Figure 2 visualises this process – for simplicity, the chart presents the allocation negotiation of one task only.

SAN is a five-stage process, involving a single source node (Node Agent S), one of the system BAs and several of other nodes in the system (Node Agent A, Node Agent B and Node Agent C). When migrating-out a given task, NA at first sends a GetCandidateNodesRequest to BA to get with a set of candidate nodes where the task can potentially be migrated to. BA scores all its cached nodes and sends back the top fifteen to NA. Additionally, in order to help to avoid collisions, BA does not directly select only top candidate nodes, but instead selects them randomly from a node pool, where candidate node score is a weight, wherein higher scored nodes are selected more frequently. This design helps to avoid a situation where an identical subset of candidate nodes is repeatedly selected for a number of tasks with the same resource requirements.

Upon receiving this list, NA sends task migration requests to all of those candidate nodes (Step 3), and waits for a given time (in this case for thirty seconds) for all replies. After this time, NA evaluates all accepted task migration responses (Step 4) and orders them in relevance order (nodes with the highest score first) and then attempts to migrate a task to a target node with top score (Step 5). If target node returns an error, the source NA will pick the next target node and attempt to migrate a task there.

At each of these stages, the target node's NA might reject task migration or return an error, for example when task migration is no longer possible because the current node's resource utilisation levels have increased or because the node attributes no longer match the task's constraints. Depending on a system utilisation level, such collisions might be more or less frequent. However, they are resolved at node-to-node communication level and do not impact the system performance as a whole.

In a situation where there are insufficient candidate nodes available due to the lack of free resource levels, the BA will return candidate nodes with the 'forced migration' flag set to true.

The algorithm's five steps are explained in the following subsections, while the forced migrations feature is detailed in supplementary subsection 5.6 below.

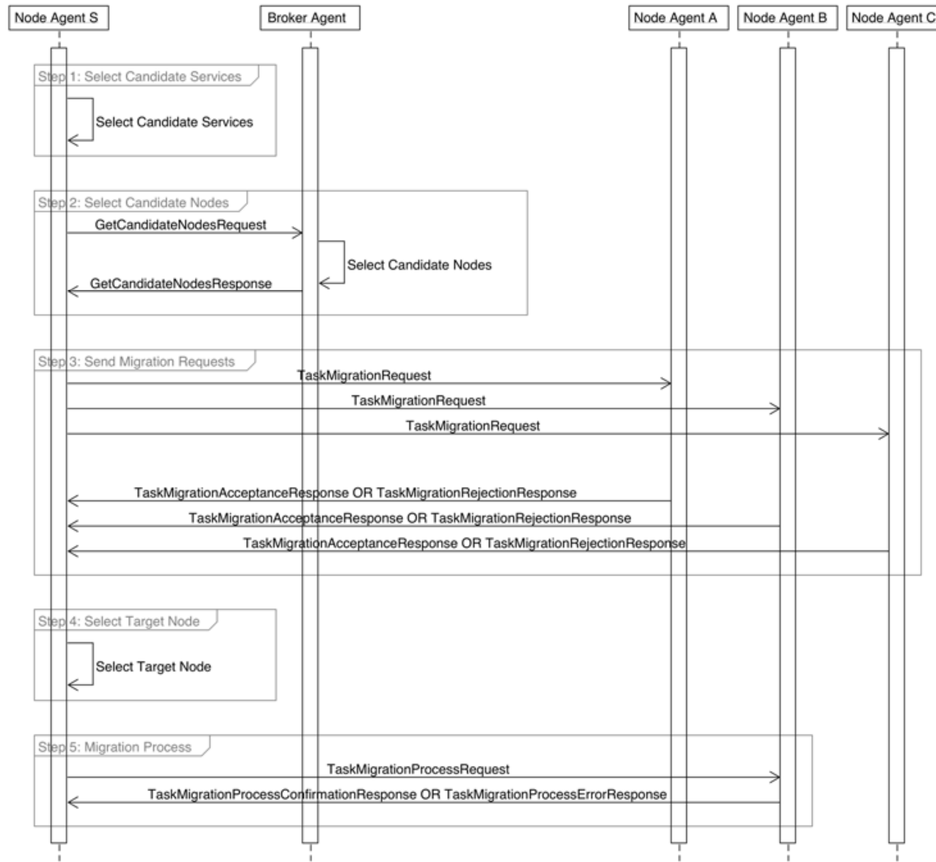

Figure 2. Service Allocation Negotiation

5.1 Step 1: Select Candidate Services

SCS routine is executed when the NA detects that the currently existing tasks are overloading its node. This step is processed on the node wholly locally. The purpose of this routine is to select the task (or set of tasks) that NA will attempt to migrate out and become stable (i.e. non-overloaded) during that process. All tasks currently running on this node are evaluated, taking into consideration various aspects, namely:

(i) The cost of running a task on this particular node. NA will aim to have the highest node score for its own node. If removing this particular task will cause its Allocation Score (AS) to be higher, then this task is more likely to be selected. AS is calculated by Service Allocation Score (SAS) functions – see section 6 for details.

(ii) The cost of migration of a task – VM migrations cause disruptions on the Cloud system. In this research, cost is estimated by Live Migration Data Transfer formula [47] as the additional network traffic required to migrate the running VM instance to an alternative node.

(iii) The likeness to find an alternative node – the majority of tasks do not have major constraints and can be executed on a wide range of nodes. However, there are a small number of tasks with very restrictive constraints that significantly limit the number of nodes that the task can be executed on. If such a task can only be executed locally, i.e. the node has enough total resources capacity and task constraints are matched, then NA is unlikely to migrate out those tasks.

(iv) Any task which cannot be executed on a local node is compulsory selected as a candidate task. This scenario could occur if the task constraints or node attributes were updated.

NA first computes a list of compulsory candidate tasks, i.e. tasks that can no longer be executed on this node. Following this, if the remaining tasks are still overloading the node, it will select a subset of tasks to be migrated out.

ISSN: **1992-8645** www.jatit.org E-ISSN: **1817-3195**

The candidate tasks selection algorithm tries to minimise the total migration cost of selected tasks, and also to achieve the highest AS for a node, under the assumption that the selected subset of candidate tasks is successfully migrated to the alternative node. In order to achieve this, the algorithm defines the Fitness Function as coded inside SCS:

$$Fitness\ Function = \frac{Node\ allocation\ score}{Total\ migration\ cost}$$

For the above in a NP-Hard problem with a substantial search space, e.g. twenty tasks on a node, the search space size is over one million combinations. Given this, the Full Scan approach [47] will be substantially computation-intensive. Therefore, the use of metaheuristic algorithms is justified. In previously researched scheduling concept, a variant of TS has been successfully applied to solve a similar class of problems. The TS algorithm has the following properties:

- It has a small memory imprint since only the list of visited solutions is maintained thorough execution;
- It can be easily parallelised as a variant which is restarted multiple times;
- It is very controllable through setting up a limited number of steps and number of runs;
- It is stoppable, and the best-found result can be retrieved immediately.
- It generally returns good results.

It was found that multiple restarts (herein a twenty-five re-run limit) with a shallow limit of steps (herein five) yield very good results, with only about 2-7% of solutions in the whole search space (i.e. selecting a subset of tasks being run on a node) being examined in each invocation. Additionally, instead of restarting the algorithm an arbitrary number of times, a stop condition for this algorithm has been implemented when the best-found solution has not been improved in a certain number of the last steps (herein six).

A sample log entry is presented below, wherein the subset of candidate tasks is being computed:

```
12:44:22.016 NodeAgentActor (node=2274790707) INFO
SAMPLE:
 Selected overloading tasks for node [2274790707]
 Node total resources = [0.5,0.2493]
 Node used resources (all tasks) = [0.5598619,0.206038]
 Node used prod resources (all tasks) = [0.481296,0.219028]
 All tasks (* Selected):
  Task [2902878580-1081] (PROD) Priority=11 Required resources=[0.006248,0.001457]
   Used resources=[0.01498,0.02692] Migration cost = 6876.02 [MB]
  Task [2902878580-3147] (PROD) Priority=11  Required resources=[0.006248,0.001457]
   Used resources=[0.01053,0.02509] Migration cost = 3820.05 [MB]
  Task [3998352223-38] (PROD) Priority=9  Required resources=[0.3125,0.1592]
   Used resources=[0.168,0.07617] Migration cost = 69139054863.11 [MB]
  Task [5726057648-7] (PROD) Priority=9  Required resources=[0.0625,0.007767]
   Used resources=[0.01682,0.005814] Migration cost = 106.72 [MB]
* Task [6218406404-243] (PROD) Priority=0  Required resources=[0.04065,0.02069]
   Used resources=[0.005684,0.005798] Migration cost = 106.69 [MB]
  Task [6218406404-959] (PROD) Priority=0  Required resources=[0.04065,0.02069]
   Used resources=[0.008255,0.005791] Migration cost = 106.67 [MB]
* Task [6251414911-1447] Priority=1  Required resources=[0.0625,0.0318]
   Used resources=[0.0007629,0.007675] Migration cost = 112.37 [MB]
  Task [6251664479-137] (PROD) Priority=2  Required resources=[0.0125,0.007767]
   Used resources=[0.04224,0.005592] Migration cost = 106.25 [MB]
  Task [6251784940-1615] Priority=2  Required resources=[0.02499,0.02545]
   Used resources=[0.02917,0.0135] Migration cost = 183.40 [MB]
  Task [6251787910-686] Priority=2  Required resources=[0.02499,0.03339]
   Used resources=[0.0321,0.01501] Migration cost = 236.79 [MB]
* Task [6251803864-88] Priority=2  Required resources=[0.02499,0.02545]
   Used resources=[0.1665,0.01027] Migration cost = 128.94 [MB]
* Task [6251812952-159] Priority=2  Required resources=[0.02499,0.07959]
   Used resources=[0.06482,0.008408] Migration cost = 115.72 [MB]
  Task [6251812952-2072] Priority=2  Required resources=[0.02499,0.07959]
   Used resources=[0,0] Migration cost = 101.00 [MB]
 Node used resources (remaining tasks) = [0.322095,0.173887]
 Node used prod resources (remaining tasks) = [0.440646,0.198338]
 Total migration cost (selected tasks) = 463.71561966381125 [MB]
```

Here, the thirteen tasks are being executed on node '2274790707'. However, the used resources exceed the node's total resources, i.e. all tasks are utilising 0.5598619 CPU, while the node can provide only 0.5 CPU (values are normalised). The node's NA detects the node is overloaded and triggers the SCS routine. The SCS routine selects four tasks (here: the production task '6218406404-243' and non-production tasks: '6251414911-1447', '6251803864-88' and '6251812952-159'; marked with *) which are then added to candidate tasks, and NA will attempt to migrate out this set in the next step.

The potential reduction of used resources is an effect of removing a subset of tasks from this node: (i) CPU reserved for production tasks is potentially reduced from 0.481296 to 0.440646 which is ca. 88% utilisation of total 0.5 CPU available on this node, and (ii) memory reserved for production tasks is potentially reduced from 0.219028 to 0.198338 which is ca. 80% utilisation of the total 0.2493 memory available on this node. The total migration cost for this set of migrations is ca. 463.72MB.

5.2 Step 2: Select Candidate Nodes

After selecting candidate nodes, NA sends a GetCandidateNodes request to BA. A part of this request, task information data, such as currently used resources and constraints, are sent. BA also itself caches a list of all nodes in system with their available resources and attributes. Based on this information, BA prepares a list of alternative candidate nodes for a task in request. The main objective of this process is to find alternative nodes which have the potentially highest node AS, under the assumption that the task will be migrated to a scored node. The size of this list is limited to an arbitrary value to avoid network congestion when NA will send actual migration requests query in the

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

next step. In this implementation, it is set to fifteen candidate nodes returned in each response.

This step is the most computing intensive of all, and represents a potential bottleneck for negotiating logic processing. BA needs to examine all system nodes, check their availability for a given task and score them accordingly. The request processing is self-contained and highly concurrent, meaning that the node scoring can be run in parallel and the final selection of top candidate nodes is run in sequence. Originally, this code was extensively profiled and improved, and designed BA to be able to run in a multi-instance mode if needed and to handle heavy usage. However, in experiments, the quoting mechanism proved to be very lightweight and the demand not that high, meaning that a single BA was sufficient to handle 12.5k nodes in the system.

Below, a sample log entry is presented when such a list is computed and returned to a NA:

```
17:53:28.516 NodeAgentActor (node=97967489) INFO
SAMPLE:
 Candidate nodes recommendations for migration-out of task:
  Task [6251414911-740] Priority=1  Required resources=[0.0625,0.0318]
  Used resources=[0.04761,0.009735] Migration cost = 124.29 [MB]
 Source node: Node [97967489] [0.5,0.4995]:
  CandidateNodeRecommendation[nodeId=2110696959,nodeAvailableResources=[0.11670008,
   0.057892],fitnessValue=5.02779735207,forceMigration=false]
  CandidateNodeRecommendation[nodeId=2274669582,nodeAvailableResources=[0.0846342,
   0.131113],fitnessValue=4.351440488446,forceMigration=false]
  CandidateNodeRecommendation[nodeId=294847211,nodeAvailableResources=[0.20732303,
   0.023297],fitnessValue=3.990484728735,forceMigration=false]
  CandidateNodeRecommendation[nodeId=1302354,nodeAvailableResources=[0.21478553,
   0.071508],fitnessValue=3.36826714248,forceMigration=false]
  CandidateNodeRecommendation[nodeId=7246234,nodeAvailableResources=[0.3283863,
   0.020561],fitnessValue=2.444197290198,forceMigration=false]
  CandidateNodeRecommendation[nodeId=2887932822,nodeAvailableResources=[0.30516457,
   0.109883],fitnessValue=2.147161970183,forceMigration=false]
  CandidateNodeRecommendation[nodeId=38743543,nodeAvailableResources=[0.3583948,
   0.101834],fitnessValue=1.769829840087,forceMigration=false]
  CandidateNodeRecommendation[nodeId=656811,nodeAvailableResources=[0.23940511,
   0.26298],fitnessValue=1.711800790297,forceMigration=false]
  CandidateNodeRecommendation[nodeId=38709566,nodeAvailableResources=[0.3584505,
   0.118908],fitnessValue=1.697701710745,forceMigration=false]
  CandidateNodeRecommendation[nodeId=3739348304,nodeAvailableResources=[0.23673398,
   0.268172],fitnessValue=1.696017579836,forceMigration=false]
  CandidateNodeRecommendation[nodeId=1093461,nodeAvailableResources=[0.380115,
   0.08412],fitnessValue=1.681960083254,forceMigration=false]
  CandidateNodeRecommendation[nodeId=4217347623,nodeAvailableResources=[0.36352026,
   0.146784],fitnessValue=1.553194840995,forceMigration=false]
  CandidateNodeRecommendation[nodeId=16918689,nodeAvailableResources=[0.3916948,
   0.125088],fitnessValue=1.456396783346,forceMigration=false]
  CandidateNodeRecommendation[nodeId=25749509,nodeAvailableResources=[0.0367722,
   0.073692],fitnessValue=0.000000000001,forceMigration=true]
  CandidateNodeRecommendation[nodeId=38679534,nodeAvailableResources=[0.38115265,
   0.006653],fitnessValue=0.000000000001,forceMigration=true]
```

Here, NA on node '97967489' requested candidate nodes for the migration of the task '6251414911-740'. BA returned top candidate nodes for a given task ordered by their suitability score, i.e. fitness value. Here values returned are: 5.02779735207 for node '2110696959', 4.351440488446 for node '2274669582', 3.990484728735 for node '294847211', 3.36826714248 for node '1302354', and so on. Additionally, the last recommendations for nodes '257495090' and '38679534' are forced-migrations (forceMigration is set to true).

Within the node recommendation there is additional information, such as node available resources and other metadata (not shown in listing). It is not necessary to return this extra information, but it was found to be very useful for logging and sampling purposes, and then efficient tuning of the system (for details see subsection 7.4).

5.3 Step 3: Send Migration Requests

Forced migration candidates will be always added to the list of accepted candidate nodes in the next step but with minimal scores. Each NA analyses its own node availability for a given task, i.e. both the available resources and the node's attributes, and responds with TaskMigrationAcceptanceResponse or TaskMigrationRejectionResponse.

Acceptance response only implies the readiness to accept a task with NA not yet allocating any resources (the resources allocation is part of task migration request process as detailed in Step 5). Additionally, TaskMigrationAcceptanceResponse message contains this node's current resources usage levels, which are used in the next step to rescore this node, since the data from BA are less recent.

5.4 Step 4: Select Target Node

NA waits for a defined time, or until all candidate nodes have responded by either the acceptance or rejection of a migrated task, and computes a list of nodes that accepted this task. NA evaluates each of the accepting nodes using the Service Re-allocation Score (SRAS) function, with the assumption that the task will be re-allocated to a scored node. From this pool, a target node is then selected. The selection is weighted with node scores but still randomised, which helps to avoid conflicts when many task migrations compete for the same node.

As noted above, all forced migration candidate nodes will be added to this list but will be selected only in last place, once all other alternative migrations attempts fail. This strategy ensures that NA always has an alternative node to offload the task. A scenario in which only one node is capable of running a given task is considered to be an error, and is reported to the system administrator. For fault-tolerance reasons, the system should always have multiple nodes able to run any given task.

A sample log entry is presented below:

```
17:48:51.541 NodeAgentActor (node=30790115) INFO
SAMPLE:
 Accepted recommendations for migration-out of task:
  Task [4844000327-3] (PROD) Priority=10  Required resources=[0.0625,0.003109]
   Used resources=[0.003742,0.001886] Migration cost = 101.86 [MB]
 Source node: Node [30790115] [0.5,0.2493]
 All non-expired recommendations (* selected):
  CandidateNodeRecommendation[nodeId=72,nodeAvailableResources=[0.24092502,0.080235],
   fitnessValue=2.737788312063,forceMigration=false]
  CandidateNodeRecommendation[nodeId=499530475,nodeAvailableResources=[0.2017312,
   0.133636],fitnessValue=2.704122369764,forceMigration=false]
  CandidateNodeRecommendation[nodeId=6608641,nodeAvailableResources=[0.15529385,
   0.18712],fitnessValue=2.657728011619,forceMigration=false]
  CandidateNodeRecommendation[nodeId=336053478,nodeAvailableResources=[0.2798536,
   0.047955],fitnessValue=2.558664832112,forceMigration=false]
  CandidateNodeRecommendation[nodeId=351638129,nodeAvailableResources=[0.212140724,
   0.146822],fitnessValue=2.505822852307,forceMigration=false]
  CandidateNodeRecommendation[nodeId=431038304,nodeAvailableResources=[0.3267638,
   0.039748],fitnessValue=2.142784872639,forceMigration=false]
* CandidateNodeRecommendation[nodeId=3650320528,nodeAvailableResources=[0.31184762,
   0.073969],fitnessValue=2.101080438228,forceMigration=false]
  CandidateNodeRecommendation[nodeId=351664198,nodeAvailableResources=[0.3099791,
   0.111418],fitnessValue=1.926755718413,forceMigration=false]
  CandidateNodeRecommendation[nodeId=656551,nodeAvailableResources=[0.3613202,
   0.1106346],fitnessValue=1.564594925411,forceMigration=false]
  CandidateNodeRecommendation[nodeId=1273895,nodeAvailableResources=[0.3402396,
   0.148566],fitnessValue=1.556209187067,forceMigration=false]
  CandidateNodeRecommendation[nodeId=662212,nodeAvailableResources=[0.40326971,
   0.065803],fitnessValue=1.431851646113,forceMigration=false]
  CandidateNodeRecommendation[nodeId=1272936,nodeAvailableResources=[0.34438919,
   0.267601],fitnessValue=1.119583713082,forceMigration=false]
  CandidateNodeRecommendation[nodeId=2594787,nodeAvailableResources=[0.3313337,
   0.363716],fitnessValue=0.874210901828,forceMigration=false]
  CandidateNodeRecommendation[nodeId=2098371268,nodeAvailableResources=[0.33264115,
   0.052842],fitnessValue=0.000000000001,forceMigration=true]
  CandidateNodeRecommendation[nodeId=1332336,nodeAvailableResources=[0.25883595,
   0.325499],fitnessValue=0.000000000001,forceMigration=true]
```

Here, NA on a node '30790115' is selecting a target node for the migration of task '4844000327-3' (with the migration cost of 101.86MB). All accepted recommendations from previous step (within thirty seconds) or forced recommendations (forceMigration is set to true) are re-scored and a single node is selected (here: node '3650320528'; marked with *). Then, NA sends TaskMigrationProcessRequest to initiate a task migration process itself. NA stores received candidate node recommendations in its memory in case the task migration fails, and the next target node has to be selected.

Once the task is removed from a node, meaning it is re-allocated, and has finished its execution, is killed or crashes, all its candidate node recommendations are automatically invalidated and deleted. Additionally, candidate node recommendations expire after an arbitrary defined time, in this case three minutes. This mechanism exists in order to remove recommendations with out-dated node data. If no candidate node recommendations are left (or expire), and the node is still overloaded, the SAN process restarts from Step 1.

5.5 Step 5: Migration Process

Every NA is actively listening for coming migration requests. When NA receives TaskMigrationProcessRequest, it performs a final suitability check, wherein both node's available resources and task constraints are validated. If the forced-migration flag is set, NA ignores the existing tasks and validates the required resources against total node resources. Occasionally, the target NA can reject task migration process or migration fails. In such a scenario the algorithm returns to Step 4 and selects the next candidate node (via weighted randomised selection).

In practice, this happens only for 6-8% of all task migration attempts (in simulated GCD workload), the majority being the result of task migration collisions where two or more tasks are being migrated to the same node. The first-to-arrive TaskMigrationProcessRequest is generally successful, meaning that Steps 4 and 5 are repeated only for the rejected migrations. There have been no observations of an increase in collisions when the larger Cloud system is simulated (up to 100k nodes, as detailed in section 7.10). This is because a single NA communicates with only a limited set of other agents, and the P2P communication model is used exclusively. This means that the communication overhead does not go up when the system size is increased.

5.6 Forced Migration

In rare circumstances, approximately 10-15 out of 10k tasks present constraints which restrict the execution of a task to a very limited number of nodes. Considering this, there is a scenario in which NA wants to migrate out a given task but is unable to find an alternative node because all suitable nodes have already been allocated to other tasks, and the majority of their resources have been utilised. In such a scenario, BA returns candidate node recommendations with a forced-migration flag set. In response, the BA can also mix non-forced migrations and forced migrations. In a worst-case scenario, all returned recommendations would be forced, but this approach ensures there is always an acceptable node to run a given task on. This prevents a starvation of the task resources, where the task is never executed.

A forced migration flag signals that a node is capable of executing a task but that its current resources utilisation levels do not allow it to allocate additional tasks, since this will cause the node to be overloaded. Forced migration forces the node to accept the task migration request while skipping the available resources check. However, task constraints are still validated, including the check if the node's total resources are sufficient to run the task. This design helps to avoid a situation where a task has very limiting constraints and only a few nodes in the system can execute it. If those nodes have no available resources then it will not be possible to allocate a task to them, and therefore tasks will not run. As such, the nodes are forced to accept this task, which then many trigger the target node's NA to migrate out some of its existing tasks to alternative nodes.

6. SERVICE ALLOCATION SCORE

SAS functions are a crucial part of the system, which greatly impacts global resource usage level. That is, they determine how well nodes' resources are utilised. They are used when a new task is allocated or when a system needs to re-allocate an existing task to an alternative node.

SAS functions evaluate how well a given task will fit a scored node system-wise by returning AS value. In this implementation, SAS input is constructed from the total node resources, the currently available node resources and the currently required resources for a given task. SAS function returns a value when a task fits the available resources on a node, and also when a node is overloaded by a task. If a node cannot fulfil a task's constraints, the node is deemed non-suitable and the scoring function is undefined. This research concludes that node AS are failing in six separate areas:

(i) Idle Node – a completely idle node is a special case of allocation, in which no task has been allocated to this node. Such a node could be completely shut down, resulting in lower power usage for a cluster. In this research, idle nodes are scored most highly when determining a suitable node for initial task allocation.

(ii) Super Tight Allocation (STA) – where some of the node's resources are utilised in the 90%-100% range. STA is regarded as stable allocation; however, due to the dynamic resource usage, this is actually not a desirable scenario. Complete, or almost complete, resource usage can frequently lead to resource over-allocation, whereby one or more tasks increase their resource utilisation. This experimentation has determined that leaving 10% of any given resource unutilised gives the best results since it reduces task migration but still ensures the efficient use of the system resources (see discussion in subsection 6.4).

(iii) Tight Allocation (TA) – where all node resources are utilised in the 70-90% range. This is the most desirable outcome as it promotes the best fitting allocation of tasks and, therefore, low resource wastage.

(iv) Proportional Allocation (PA) – while tight-fit is the most desirable outcome, the majority of tasks in this research consumed a small amount of each resource. Most scheduled tasks are short batch jobs which have a very short execution time. In such a scenario, it is desirable to keep proportional resources' usage ratios on all nodes which would, therefore, generally enable nodes to fit more tasks with ease.

(v) Disproportional Allocation (DA) – where the node's resources are not proportionally utilised, thereby making it difficult to allocate additional tasks if required. For example, a setup where tasks on a node allocate 75% of CPU but only 20% of memory is not desirable.

(vi) Overloaded Node – when allocated resources overload the total available resources on the node. Naturally, this is an unwanted situation, and such a node is given a score of zero.

Several types of resources exist which can be utilised by the task, such as memory, CPU cycles and disk I/O operations, and so on. The model also supports artificial resources, called 'virtual resources' and the number of defined resources is potentially unlimited. Figure 3 visualises AS types for the two resources (CPU and memory):

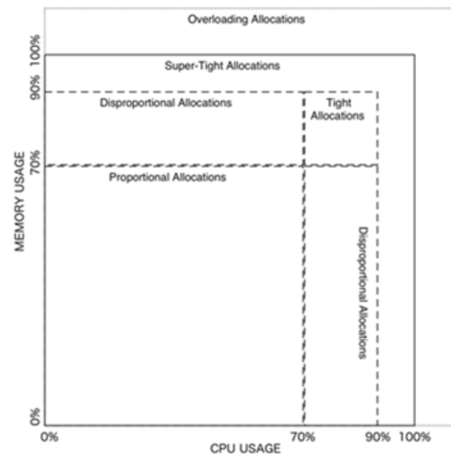

Figure 3. Allocation Score types

SAS function should never allow overloading allocations to take place in order to prevent a scored node to become overloaded and unstable. Additionally, during the research it was determined that STAs are very prone to over-allocate nodes and are damaging to overall system stability. Therefore, they are also accorded a score of zero. DAs increase global resource wastage and should be avoided; nevertheless, they are acceptable if none of the more desired types of AS are possible. The desirability order varies and depends on the task's state, as discussed in subsections 6.2 and 6.3 below, while the following subsection introduces the concept of Service Allocation Lifecycle (SAL).

6.1 Service Allocation Lifecycle

Tightly fitting tasks on as few nodes as possible are beneficial for global system throughput.

Journal of Theoretical and Applied Information Technology
31st October 2018. Vol.96. No 20
© 2005 – ongoing JATIT & LLS
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

However, during this research the following facts were observed:

(i) Initially, a Cloud user specifies the task's required resources. Users tend to overestimate the amount of resources required, wasting in some cases close to 98% of the requested resource [33]. Therefore, only after the task is executed could realistic resource utilisation values be expected. Allocating new tasks in a tight-fit way (i.e. TA and STA areas in Figure 3) does result in turmoil when the task is actually executed and the exact resource usages levels are logged. Therefore, the initial allocation should rather aim to distribute tasks across nodes and keep the resource utilisation levels on individual nodes low (i.e. PA area in Figure 3), than pile them on the lowest possible number of nodes.

(ii) In GCD, only about 20-40% of tasks qualify as long-running tasks, meaning that they run for longer than twenty minutes [44]. The remaining scheduled tasks consisted of short-term jobs which generally have much lower resource requirements than long-running tasks. The majority of tasks are short and will not exist for long at all in the system. Therefore, it is important for an initial allocation not to spend too much time in trying to tightly fit them into available nodes.

(iii) While the majority of tasks are short-lived (up to twenty minutes), there exists a number of long-running tasks that have more demanding resource requirements, meaning that the majority of resources (55–80%) are allocated to long-lived services (ibid.). Therefore, it is more difficult to fit them into nodes, and these allocations should be much tighter to minimise global system resource waste.

Given the above reasons, the ideal scenario for a task is to be initially allocated on a lowly-utilised node, before it is gradually migrated towards more tightly-fitted allocations with other tasks.

Originally, the MASB framework did not have distinct scoring functions for Service Initial Allocation Score (SIAS) and SRAS; a single SAS function, with the same scoring model as SRAS, was used for all allocations which resulted in lowered performance. The design was ultimately altered, and SAS function was split:

(i) During Initial Allocation, a randomly selected BA is responsible for allocating a newly arrived task to a worker node. BA uses SIAS function (detailed in subsection 6.2) to score nodes. Only a limited number of candidate node recommendations are calculated (here: 200) before selecting the top recommendations. This is to prevent scoring routine calculations from processing for too long. The limit of 200 applies only to non-forced recommendations for matching nodes.

(ii) A once allocated (and running) task can be re-allocated to an alternative node if necessary. In such a scenario NA of a node which the task is being executed is responsible for finding a candidate node. Both NA and BA use SRAS function (detailed in subsection 6.3) to score candidate nodes. Similar to calculating recommendations for new tasks, as an additional optimisation, only a limited number of candidate node recommendations are calculated before selecting the top recommendations. However, because this routine is invoked much less frequently, two thousand nodes are analysed. The two thousand limit applies only to non-forced recommendations for matching nodes.

MASB uses a network of BAs to provide a set of the best candidate nodes (nodes with the highest AS) to allocate the task. However, some applications such as Big Data frameworks often send multiples of an identical task in a batch. Those tasks execute the same program and have the same (or very similar) resource requirements. As such, a limited set of nodes will be highly scored and may result in a multiple repeated allocations requests to the same node over a very short period of time. To prevent this phenomenon, the pool of candidate nodes is randomly shuffled each time BA receives a request.

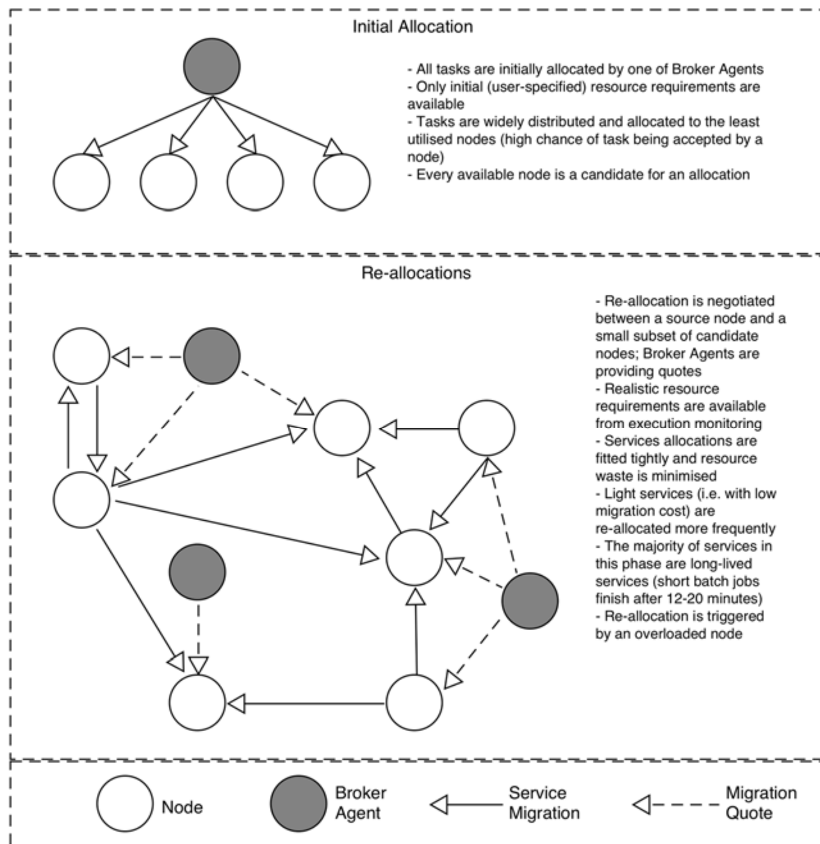

Figure 4. Service Allocation Lifecycle

6.2 Service Initial Allocation Score

As explained in the subsection above, in order to minimise the impact of Cluster user's overestimating resource requirements, the initial allocation should attempt to spread tasks widely across all system nodes. Therefore, when initially allocating existing tasks, candidate nodes should be scored in the following order: PA, TA and finally DA.

In this implementation, the SIAS function for two resource types (CPU and memory) was used. Figure 5 is a graphical representation of SIAS function:

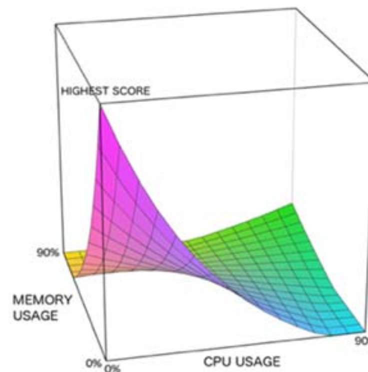

Figure 5. Service Initial Allocation Score

Three separate areas can be noticed:

(i) Lower-left (the highest score) – this promotes PA, which will leave resource utilisation at a low level or proportionately used.

(ii) Upper-right corner (the medium score) – this promotes TA, where tasks on this node will closely utilise all its resources.

© 2005 – ongoing JATIT & LLS
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

(iii) The upper-left and lower-right corners (the lowest score) – these DAs will leave one resource utilised almost fully and the other resource wasted.

It should be noted that the maximum resource usage is 90%, and that values above this level are in an undesired STA's area (and have zero AS). The following SIAS function was used:

$$SCORE = F_STEEP^{(r_{CPU} - F_BIAS \cdot r_{CPU_MAX}) \cdot (r_{MEM} - F_BIAS \cdot r_{MEM_MAX})} - F_FLOOR$$

- r_{CPU}, r_{MEM} – current resources utilisation levels on a node (values are normalised to between 0 and 1);
- r_{CPU_MAX}, r_{MEM_MAX} – total resources available on a node (values are normalised to between 0 and 1);
- F_BIAS – score factor which sets the bias towards low (i.e. SIAS function) or high (i.e. SRAS function) utilisation of resources on a node. Here, a value of 0.3 was used;
- F_STEEP – parameter describing how aggressively the system should increase scores of the more desired AS-es (which impacts the probability of a node selection). Here, a value of 350 was used;
- F_FLOOR –parameter describing how aggressively the system should reduce scores of less desired AS-es (which impacts the probability of skipping a node). Here, a value of 0.8 was used;

Additionally, negative score values are adjusted to zero (to prevent the selection of a node). It should be noted that the SIAS is calculated exclusively from user-defined resource requirements since the actually-used resource requirements are unknown before the task execution actually starts.

6.3 Service Re-allocation Score

This research has found that the best throughput results are achieved when tasks are packed tightly into available nodes, i.e. where global resource utilisation is the highest. The best fit scenario, where the task fully utilises 90% of all available resources on a node, is scored the highest. Therefore, when migrating existing tasks, candidate nodes should be scored in the following order: TA, PA, then DA.

Like the SIAS function presented in 6.2, the SRAS function for two resource types (CPU and memory) was used. Figure 6 is a graphical representation of SRAS function:

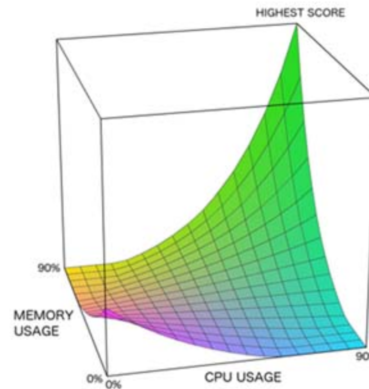

Figure 6. Service Re-allocation Score

Three separate areas can be noticed:

(i) Upper-right corner (the highest score) – this promotes TA, where tasks on this node will closely utilise all its resources.

(ii) Lower-left (the medium score) – this promotes PA that will leave resource utilisation at a low level or proportionately used.

(iii) The upper-left and lower-right corners (the lowest score) – these DAs will leave one resource utilised almost fully and the other resource wasted.

In this implementation, the following SRAS was used:

$$SCORE = F_STEEP^{(r_{CPU} - F_BIAS \cdot r_{CPU_MAX}) \cdot (r_{MEM} - F_BIAS \cdot r_{MEM_MAX})} - F_FLOOR$$

with the exceptions of F_STEEP where a value of 500 was used and F_BIAS where a value of 0.6 was used; the parameter definitions are the same as in SIAS function in subsection 6.2)

As can be observed visually, SRAS is a mirror image to the SIAS function (presented in Figure 5). The main difference is changing the score bias (i.e. F_BIAS parameter) which shifts the peak score point from (0,0) to (90,90) (percentage of utilised resources), and which relates to the change in the most desirable AS from PA to TA.

It should be noted that the SRAS is calculated exclusively from actually-allocated resource requirements. User-defined resource requirements are evaluated as part of the Resource Usage Spikes (RUS) routine, explained in detail below.

6.4 Resource Usage Spikes

Occasionally, a task might instantly increase its resource usage as the result of sudden increase of a demand for a task; at such times, a node should have the capacity to immediately accommodate this request, without needing to migrate the task to an

alternative node (since this takes time). In such a situation, other VMs running on this machine can be paused or killed to let the VM instance executing this task instantly allocate more resources.

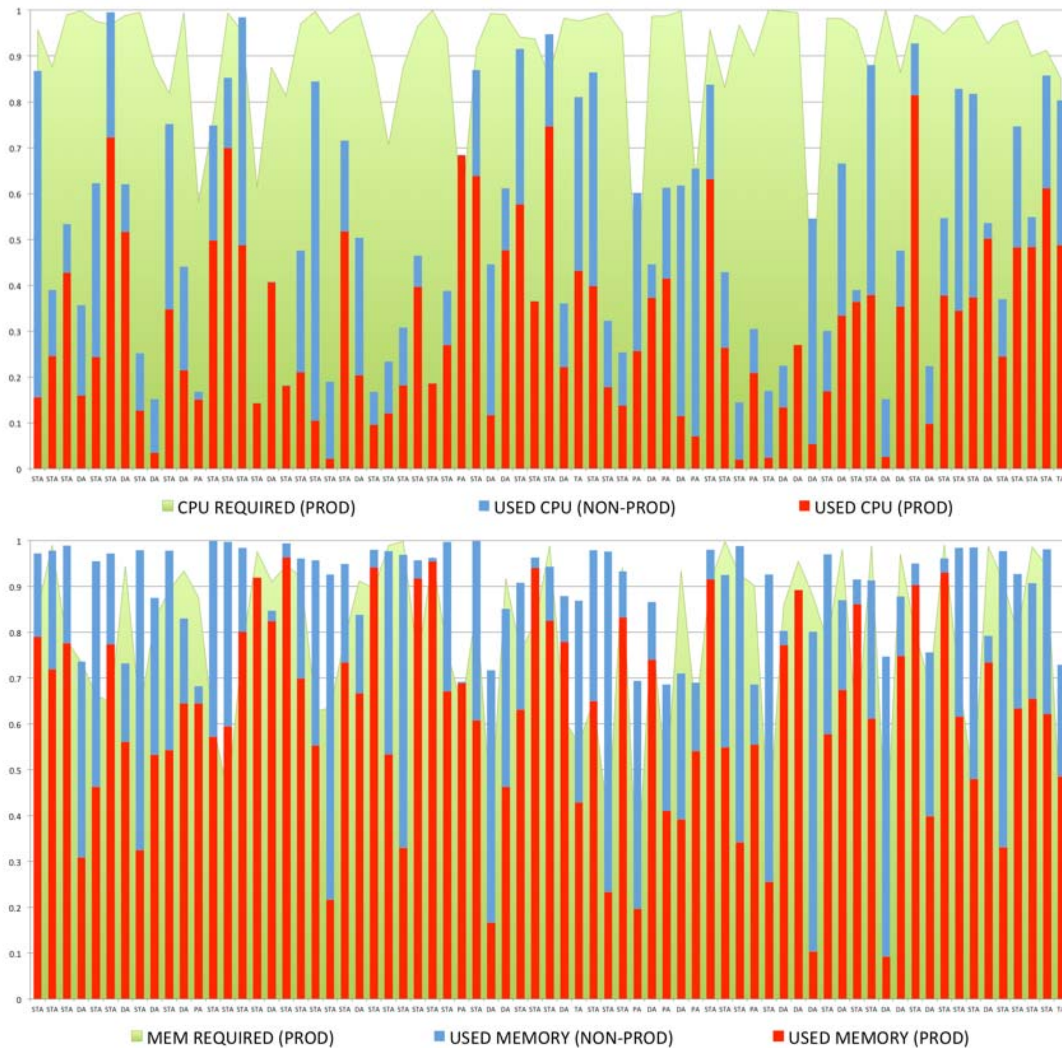

Figure 7. Production vs. non-production allocated resources

As such, an additional feature was implemented in MASB to handle RUS. Aside from checking the actually-used resources for tasks and ensuring that the node has the capacity to support it, the system also calculates the maximum possible resource usage of all production tasks based on user-defined resource requirements, as well as making sure that the node has the capacity to support all production tasks at their full resource utilisation. This constraint is limited only to production jobs since VMs running non-production jobs can be suspended without disturbing business operations.

The introduction of RUS constraint adds another dimension to the tasks allocations' logic. Figure 7 visualises how user-defined resource requirements for production tasks and actually-used resources for all tasks are integrated. In this 60-node sample (a single bar represents one node), approximately half the nodes have a very high CPU user-defined allocation for production tasks, while the real usage is much lower. It should be noted that while memory usage stays proportionally high thorough the GCD workload, the gaps between the requested and the actually-used memory are much smaller. This is a relatively common pattern for GCD workload.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Whilst RUS do not occur frequently, they do have the significant potential to destabilise an affected node. Table 2 represents the average frequency of RUS in examined GCD workload traces with ca. 12.5k nodes and ca. 140k tasks being continuously executed by them (with different RUS thresholds examined):

RUS threshold	Average RUS (count per minute)	Peak RUS (count per minute)
5%	659	7538
10%	212	4362
15%	66	2390
20%	47	1925
25%	26	1135

Table 2. Resource Usage Spike frequencies

Here, while running a simulation based on replaying the original Borg's allocation decisions (as detailed in subsection 7.7), the RUS threshold of 10%, i.e. whenever there was a greater than 10% increase in the overall node resource utilisations levels in any of the monitored resources, was breached 212 times per minute on average, with a peak of 4362 breaches.

In this research, a threshold of 10% was selected for the experimental simulations as an overall good balance between efficiently allocating nodes' resources and, at the same time, leaving the running tasks enough headroom for occasional activity spikes. Generally, lower thresholds resulted in many task migrations (and thus incurred additional task migration costs), and the thresholds above 10% were not utilising resources effectively (the system throughput was lowered). Consequently, the SAS functions were tuned to allocate up to 90% of all available resources on the node (as seen in Figure 3) which seem to give the best overall results.

RUS are a significant design consideration, and a misconfiguration might lead to multiple premature terminations of the tasks and suboptimal performance of the system. Google's engineers implemented a custom resource reservation strategy using a variant of step moving average, as detailed by John Wilkes in a presentation during the GOTO 2016 conference in Berlin [55].

7. EXPERIMENTAL RESULTS

The previously developed AGOCS framework was used as the base of the experimental simulation. AGOCS is a very detailed simulator which provides a multiple of parameters and logical constraints for simulated jobs. The scope of the available variables is very broad, including memory page cache hit and instructions per CPU cycle; however, in this project simulations were based on the following assumptions:

- Requested (by user) and realistic (monitored) resources' utilisation levels for memory and CPU;
- Detailed timing of incoming tasks and any changes in available nodes (within one-minute cycles);
- Nodes attributes and attributes' constraints defined for tasks (as specified in GCD workload traces).

This level of detail comes at the price of extensive computing power requirements. While dry simulation itself can run on a typical desktop machine, adding layers of scheduling logic, agents' states and inter-system communication requires a significant increase in processing time. In order to realistically and correctly simulate scheduling processes on a Cloud system, the Westminster University HPC Cluster was used.

7.1 Test Environment and Code Profiling

The MASB prototype was initially developed on a personal desktop, but as the size and level of detail of the simulations grew, it was necessary to move to a Cluster environment where more computing power was available. All the experiments were executed on the Westminster University HPC Cluster, regarding which more details concerning the software and hardware specifications can be found in Table 3 below:

Model	Dell R630
Operating System	CentOS Linux release 7.2.1511 (Core)
CPU	20x 2.3GHz Intel E5-2650 v3
Memory	96GB memory
Storage	1TB
Networking	10Gb Ethernet
Java Virtual Machine	OpenJDK 64-Bit Server VM (build 25.91-b14, mixed mode)

Table 3. Westminster University HPC Cluster node (March 2016)

While this cluster offered a sizable array of GPUs, the simulations did not take advantage of that computing power, and instead all processing took place on CPUs. Although it would have been possible to achieve higher throughput when using GPU with frameworks such as ScalaCL or Rootbeer, JVM does not natively support GPU processing. Having as few external dependencies as possible was therefore preferred, since they make maintaining the project more time-consuming. Interestingly, Google's BorgMaster process, which manages a single cell in the production environment for one computing cell, uses 10–14 CPU cores and up to

50GB of memory. The statistics presented are valid for an intensely utilised computing cell, for example one which completes more than 10k tasks per minute on average [51].

In experiments, MASB allocated all available forty CPU cores on HPC machine and used them continuously at 60% to 80%. The MASB process allocated ca. 7GB of memory. It is difficult to measure exactly how much computing power was spent on supporting activities such as simulating messaging interactions between agents, i.e. enqueuing and dequeuing messages to and from Akka actors. However, after tuning exercises of the default configuration, the Akka Actors framework proved to be quite resilient. It is estimated that the framework's processing did not take up more than 10-15% of the total CPU time, with the relatively lightweight AGOCS simulator framework consuming about 15-25% of all CPU time. As an interesting note, Akka's optional Thread-pool executor performed noticeably better on the test HPC machines than on the default Fork-join-pool executor, which is based on a work-stealing pattern.

However, in a truly multi-core environment, a different approach was required – one which focused on minimising context switches frequency and average CPU idle time across all available cores. Once the MASB framework was moved into the Cluster environment, the 'pidstat' command tool was used to gather statistics, before the refactor and fine-tune framework so as to achieve better parallelism. During MASB simulations, the typical observed context switches frequency was ca. 500-700 per second per thread, which is comparable with a fully loaded webserver [29].

7.2 Testable Design

Building a framework which fully simulates the Google computing cell from GCD traces has been previously recognised as a challenging task, where there are many aspects to consider [2][45][64]. GCD traces contain details of nodes, including their resources, attributes and historical changes in their values. Traces also contain corresponding parameters for tasks, such as user-defined and actually-used resources, as well as attributes' constraints. This has created a multi-dimensional domain with a range of relations which has resulted in complex error-prone implementation. In order to mitigate the risk of coding errors, especially during rapid iterations, a number of programming practices were used:

(i) A comprehensive test units suite was developed, along with prototype code. Test units were executed upon every build to catch errors before being deployed to production. This software engineering pattern allowed for a rapid development of prototype and helped to maintain the high code quality;

(ii) A number of sanity checks were built into the runtime logic, such as checking whether the task's constraints could be matched to any node's attributes within the system and checking whether the total of all scheduled tasks' resources exceeded the computing cell compatibilities;

(iii) Recoverable logic flow was implemented for both NA and BA. In the case of various errors such as division by zero or null pointer exceptions, the error is logged but the agent continues to run;

(iv) Keeping a separate error log file with the output of all warnings and errors was a considerable help in terms of resolving bugs.

The implementation of the above features gave high confidence in terms of realising a good quality and reasonably bug-free code.

7.3 Platform Outputs

Adding detailed logging features to MASB has proved surprisingly difficult. Due to the highly parallel nature of the simulated Cloud environment, an enormous number of log messages were generated upon each simulation, making it difficult to analyse the behaviour of tested algorithms. In addition, writing and flushing log streams caused pauses in simulation. Switching to a Logback framework designed with a focus on concurrent writes provided a solution to this problem, although it was necessary to split the data into distinct log files in order to improve readability, e.g. separate errors from algorithms' output data. The following outputs were used:

(i) Logging – in order to fine-tune MASB, excessive logging routines were implemented. All messages, counters and errors are logged to four types of log-files: /logs/*.log files – standard log outputs containing all logs messages and also samples; /logs/*-error.log – errors and corrupted data exceptions are written to separate files to help with debugging and troubleshooting; /logs/*-ticks.csv – CSV files with periodically generated overall system stats, such as the number of idle and overloaded nodes, number of migration attempts, global resources-allocation ratio, and so on; /usage/*.csv – detailed node usage stats and task allocations are written periodically to a file, that is, every hundred minutes of simulation time.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

(ii) Sampling – while examining every decision process in MASB simulation is virtually impossible, frequent and recurrent analysis of the details and values was useful for fine-tuning the system and the scoring functions. Not all the details of every single decision process were logged, rather just a small percentage of all invocations. In the current implementation, the following items are sampled:

- The selection of overloading tasks by the NA, ca. 1 sample per 50 invocations (a sample is presented in 5.1);
- The scoring and selection of candidate nodes by the BA, ca. 1 sample per 5k invocations (see log entry in 5.2);
- The selection of the target node from the candidate node list, ca. 1 sample per 5k invocations (as listed in subsection 5.4).

Sampling proved to be one of the most important logging features implemented.

7.4 System Evolutions and Optimisations

In order to achieve high resources utilisation and low resources waste, several enhancements were implemented and then fine-tuned, including:

(i) Limiting the number of candidate nodes returned from BA to fifteen, and introducing the forced migrations feature (subsection 5.6);

(ii) Fine-tuning SCS routine to maintain the balance between migration cost and the node allocation score, which refers to finding the right combination of steps of the TS algorithm, as well as its termination depth;

(iii) Splitting the SAS function into SIAS and SRAS and then limiting the number of candidate nodes examined in those functions (200 and 2k respectively);

(iv) Adjusting input parameters for SIAS and SRAS functions, namely values for F_BIAS, F_STEEP and F_FLOOR for the best results based on samples logged (subsections 6.2 and 6.3);

(v) Adding the timestamp parameter to the candidate node recommendations, and regularly removing those which have expired. In scenarios where the task migration request is repeatedly refused, this mechanism forces NA to disregard the results of old calculations and request newly scored recommendations from BAs. In this implementation, the recommendation's age threshold was set to three minutes (simulation time) with lower values not yielding better results (see subsection 5.4).

7.5 Test Simulations Setup

During the later stages of the development of the MASB prototype, several simulations were continuously run. They were frequently paused, tuned and then resumed to see whether a given tweak would improves the results. This methodology allowed the research to progress at a good speed while simultaneously iterating a number of ideas and tweaks. Therefore, the testing process did not have noticeable stages, but instead the stages blended into each other. This said, it is possible to logically split the testing into four main areas:

(i) Benchmarking – GCD workload traces also contain actual Google's Borg scheduler task allocations. In the Borg's simulation, MASB will replay all recorded events, mirroring tasks allocations as per the Google scheduler, i.e. not using its own scheduling logic. This simulation was used as a controlling run in order to test the system, and also as a benchmark to compare results with the original allocations.

(ii) Throughput – secondly, MASB was tested to identify whether it was capable of allocating the same workload as Borg system. The size of the workload was then increased gradually in 2% steps while preserving the configuration of the system nodes. To ensure the correctness of results, another technique, called 'cell compaction' [51] was used in which, instead of adding additional tasks, the system nodes were removed. The results were then compared to the original GCD workload.

(iii) Migration Cost – thirdly, this batch of experiments focused on migration costs incurred via use of VM-LM. A collection of different SAS functions and their variants were tried in order to research their impact on total migration cost while allocating the given workload.

(iv) Scalability – finally, the MASB simulation was run with multiplies of GCD workload in order to test the scalability limits of the designed solution. Although this step was the least work-intensive, it took the longest time to perform.

As noted in [64], simulating GCD workload is not a trivial task. The main challenge when running such large and complex simulations is the demand for computation power and the continuous processing. During this experiment, the AGOCS framework was modified to also allow the testing of computing cells larger than 12.5k. This was achieved by duplicating randomly selected existing tasks and their events, for example 'Create Task A event from GCD workload trace files' will create events AddTaskWorkloadEvent events for task A and A'.

Journal of Theoretical and Applied Information Technology
31st October 2018. Vol.96. No 20
© 2005 – ongoing JATIT & LLS
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

This feature is based on the hashcode of object's ID, which is a constant value.

The largest experiments simulated a single Cloud computing cell with 100k nodes and required nine months of uninterrupted processing on one of the University of Westminster HPC cluster's nodes. At this juncture, it should be noted that early simulations often fail due to unforeseen circumstances, such as NAS detachment or network failure. One solution to this was to frequently save snapshots of the state of the simulation and to keep a number of previous snapshots in case of write file failure.

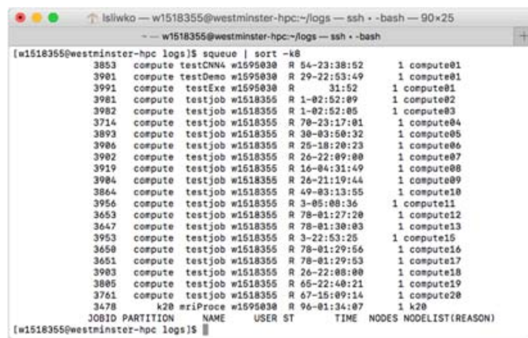

Figure 8. University of Westminster HPC Cluster utilisation

At the peak of the experiment, eighteen out of twenty computing nodes were committed to running MASB simulations, as can be seen in Figure 8.

7.6 Allocation Score Ratios

Clearly, when examining the suitability of load balancing, the key parameter is the number of overloaded nodes, which should be kept to minimum. It was found that replaying GCD traces using Google's original Borg's allocation decisions results in up to 0.5% of nodes being overloaded in a simulated one-minute period. It was assumed that this phenomenon was the result of delayed and compacted resource usage statistics, which were recorded and averaged over ten-minute periods. As such, in further experiments this ratio was used as an acceptable error margin.

The second researched property was how nodes were distributed amongst allocation score types during simulations. Therefore, each experiment recorded a number of nodes with each allocation score type, and averaged them out over the simulation period. The set of normalised values for STA, TA, PA and DA are referred to as Allocation Score Ratios (ASR). Idle Nodes and Overloaded Nodes are discussed separately, and they are excluded from the ASR. The ASR values describe how well the Cluster is balanced, that is, how well nodes are balanced as a whole group.

The ASR values are used to describe the experimental results presented in the subsections below to highlight the differences in how various load balancing strategies perform under a GCD workload.

The most dominant AS was PA, meaning that each of the node's resources is utilised between 0% and 70%. Ca. 68% of all the cluster's nodes are found within these parameters, which is the direct result of their initial allocation using SIAS function. The second biggest group, ca. 22% of all servers, are nodes allocated disproportionally in which one or more resources are highly used but the other resources are relatively idle. The remainder of the nodes have either an STA or TA allocation score type. The PA to DA ratio of roughly 3:1 is characteristic for a typical workload as recorded in GCD traces and processed by MASB.

Figure 9 chart visualises the AS distribution during a month-long simulation. The horizontal axis is the measure of time and the vertical axis represents the number of nodes having a particular allocation type (as per coloured legend). The chart also highlights two periods of low and elevated workload, marked A and B respectively:

(i) During the low workload period (A), SIAS function can schedule most newly-arriving tasks to relatively unused nodes, thereby successfully preserving their resource usage proportions. As such, the number of PAs increases while the number of DAs decreases. Existing long-running services continue to run uninterrupted on their nodes, and so the ratio of STA to TA remains flat.

(ii) During an elevated workload period (B), SIAS function is unable to find relatively unused nodes anymore. It thus selects lower quality allocations, resulting in a decrease in PAs. Due to the scarcity of resources, tasks are also re-allocated more frequently by SRAS function. This results in tighter fit allocations, which is seen as an increase in STAs and TAs counts.

This cycle is repeated thorough cluster activity, wherein MASB balances the workload. The subsections which follow describe several implemented optimisations and their rationales, as well as the experimental results and a commentary on them.

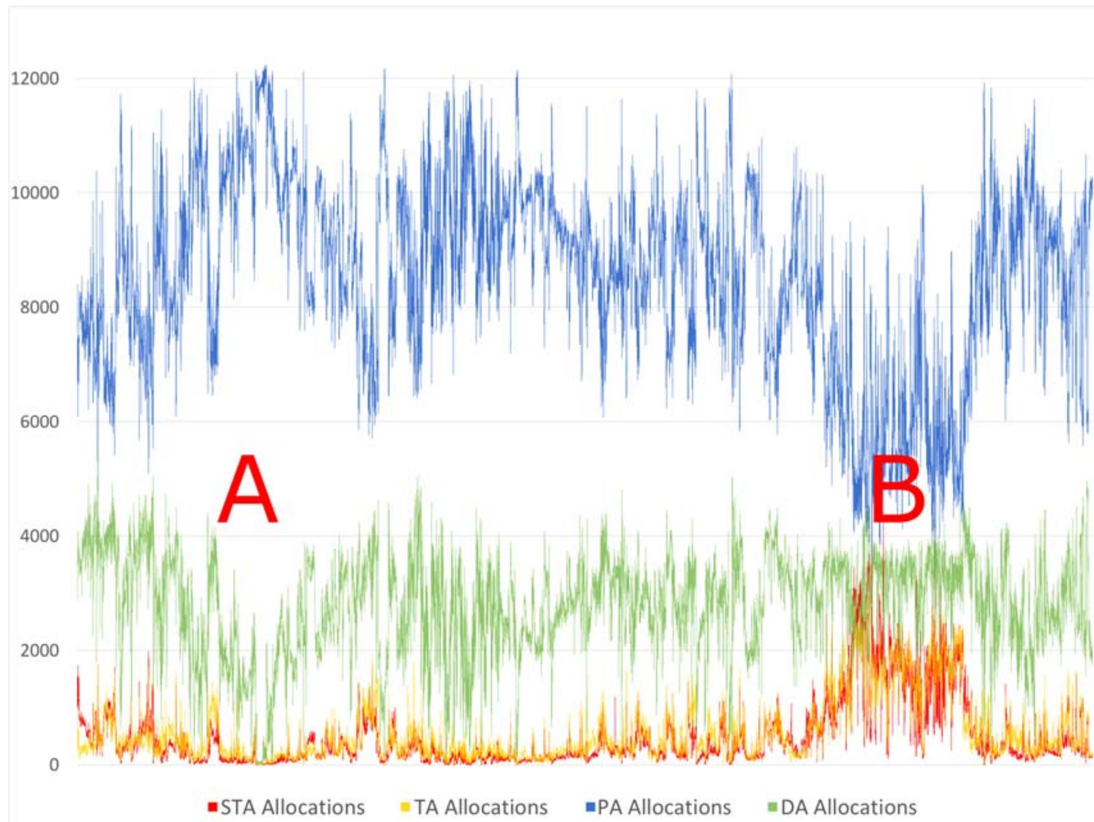

Figure 9. MASB – Allocation Scores distribution (12.5k nodes)

7.7 Benchmark

Given that GCD traces have a complicated structure and contain a vast amount of data, only rarely are they analysed to the full extent of their complexity. MASB design shares similarities with BorgMaster in areas such as constraining tasks, defining memory and CPU cores as resources, using scoring functions for candidate node selection, and handling RUS. It also closely follows the lifecycle of tasks as presented in [19]. As things stand, there is no publicly available literature which contains descriptions of similar experiments which could be compared with the simulation results of MASB. Therefore, the closest comparable results are the original Borg's allocation decisions that were recorded in GCD traces. For the purposes of this research, it was decided that they be used as a benchmark for the results from MASB's experiments.

Both simulations processed full month-long GCD traces. The average values were used because MASB simulation works in one-minute intervals whilst GCD traces provide usage statistics in ten-minute windows that occasionally overlap. Given this, peak or median values were not accurate.

To highlight differences in workings between the MASB and Google Borg algorithms, Figure 10 presents the AS distribution during the period recorded in GCD (replayed Google's Borg allocation events). In comparison to the experimental data presented in Figure 9, MASB behaves more organically during periods of low and elevated workload. This is especially visible during the period of elevated workload (B) where MASB managed to preserve a better ratio of PA to DA Nodes than Google's Borg. This behaviour is the result of allowing a given task to be re-allocated during its execution, meaning that MASB can dynamically shape its workload and improve the health of its allocations. This feature also allows greater flexibility in altering the requirements of running tasks, in which the load balancer attempts to offload an alternative node.

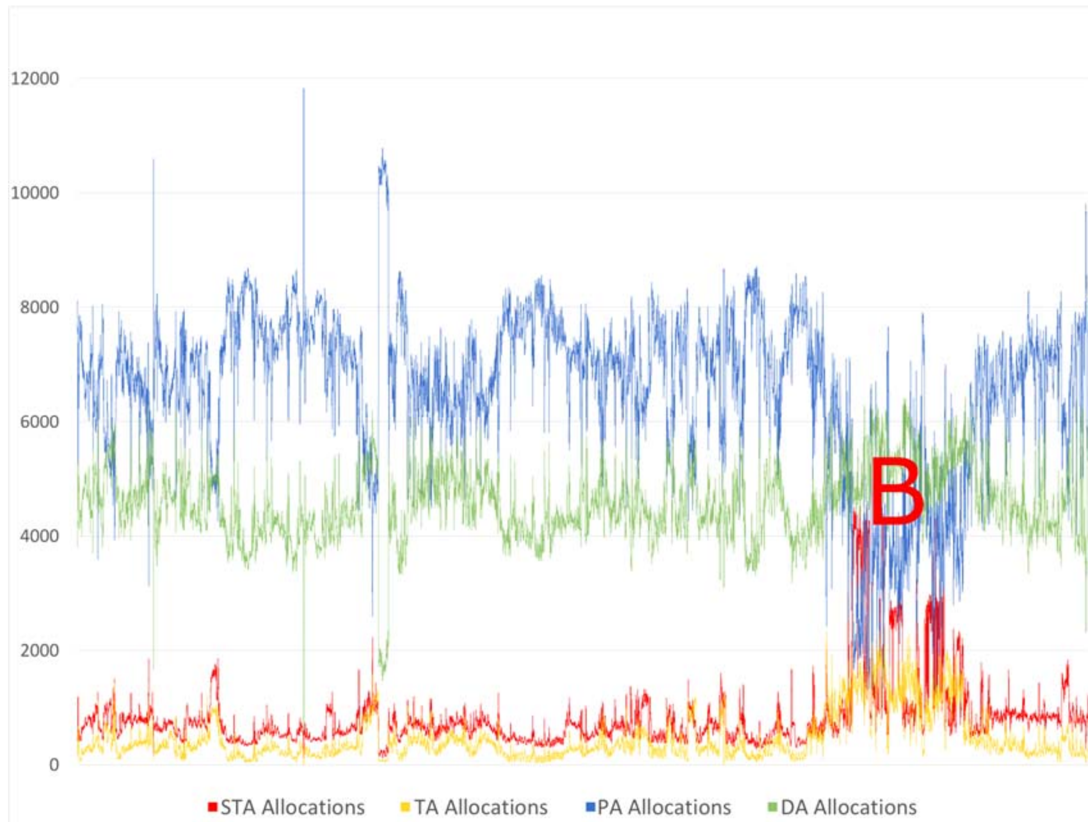

Figure 10. Borg – Allocation Scores distribution (12.5k nodes)

Table 4 directly compares ASR parameters of both pre-recorded Google's Borg and MASB simulations:

Parameter (average, one-minute interval)	Framework	
	Borg (Figure 10)	MASB (Figure 9)
Idle Nodes	1.01 (0.01%)	78.10 (0.63%)
STA[1] Nodes	820.49 (6.58%)	487.01 (3.91%)
TA[1] Nodes	459.57 (3.69%)	564.18 (4.53%)
PA[1] Nodes	6597.14 (52.94%)	8508.08 (68.28%)
DA[1] Nodes	4578.49 (36.74%)	2810.69 (22.56%)
Overloaded Nodes	4.04 (0.03%)	12.62 (0.10%)

Table 4. Benchmark results – Borg and MASB

The listed ASR values highlight the differences in Borg and MASB workings:

(i) Idle Nodes – Borg's design has a definite advantage over MASB because Borg's schedulers can access the shared cluster's state and iterate over the complete set of system nodes. MASB relies on a network of BAs, each of which has only partial information about the cluster's state. Therefore, a subset of idle nodes might never be scored, even if they represent the best allocation for a given task.

(ii) STA and TA Nodes – in both systems, under normal workload conditions, incoming tasks are reasonably well distributed between the nodes. Only ca. 10% of all system nodes register higher resource usage scores, when at least one of resource utilisation levels crosses 90%. The exact scoring algorithm of Google's Borg has not been disclosed, but the results suggest a degree of similarity to the SIAS function.

(iii) PA and DA Nodes – the ratio of PAs to DAs is visibly different in Borg and MASB. Borg's original scheduling decisions had a ratio of roughly 3:2, meaning that for every three proportionally allocated nodes in the system, there were two nodes that were disproportionately allocated. MASB managed to achieve a better ratio of 3:1, suggesting that the use of SIAS and SRAS scoring functions together with VM-LM feature can potentially create a more balanced scheduling system.

Given the superior ratio of PA to DA nodes as measured, and the possibility of increased

throughput, the next experiment focused on processing increased workload.

7.8 Throughput Tests

The MASB framework has been designed as a general solution for balancing workload in a decentralised computing system. After numerous iterations, MASB was eventually able to schedule the entire GCD workload, with additional tasks also added.

Table 5 presents a comparison of the results with different workload sizes:

Parameter (average per minute)	Workload Size (tasks)			
	100% (original)	102%	104%	106%
Nodes Count	12460.39	12460.36	12460.68	12460.35
Tasks Count	132061.15	134738.92	137399.93	142936.05
Global CPU Usage Ratio	43.64%	44.54%	45.42%	46.89%
Global Memory Usage Ratio	62.05%	63.33%	64.58%	66.57%
Idle Nodes	76.41 (0.61%)	73.08 (0.59%)	72.75 (0.58%)	52.18 (0.42%)
STA Nodes	479.91 (3.85%)	480.22 (3.85%)	423.51 (3.40%)	447.73 (3.59%)
TA Nodes	566.20 (4.54%)	545.74 (4.38%)	447.75 (3.59%)	355.88 (2.86%)
PA Nodes	8507.49 (68.28%)	8718.76 (69.97%)	9316.35 (74.77%)	9576.67 (76.86%)
DA Nodes	2818.11 (22.62%)	2610.04 (20.95%)	2084.06 (16.73%)	1718.08 (13.79%)
Overloaded Nodes	12.28 (0.10%)	32.53 (0.26%)	116.25 (0.93%)	309.79 (2.49%)

Table 5. Throughput results (100%-106% workload)

As demonstrated above, MASB was able to schedule, on average, an additional ca. 2.6k tasks per minute (ca. 2% more tasks). Further tuning was unable to improve those results, with workload sizes greater than 102% increasing the number of overloaded nodes above the defined threshold of 0.5%.

To further ensure the correctness of the attained results, another set of experiments was run in parallel. Here, instead of multiplying the original GCD workload, the random machines were removed from the cluster until the workload could no longer be fitted. This method, known as 'cell compaction', is suggested in [51] for simulations with GCD traces.

Similar to the previously detailed experiments which had augmented workload, even when the cluster size was reduced to ca. 98% of its original size (242 nodes being removed), the original GCD workload could still be fitted without breaching the 0.5% limit of overloaded nodes.

On average, GCD traces utilise ca. 40-50% of the globally available CPUs and ca. 60-70% of globally available memory while continuously guaranteeing ca. 85% of CPUs and ca. 70% of memory to production tasks to handle RUS. It should be noted that Borg's scheduling routines have been perfected following decades of work by a team of brilliant Google engineers. The conclusion of this research is that, it is hard to substantially improve this impressive result given those constraints. Although the throughput of the original Google Scheduler could not be significantly improved, the results from both methods of evaluation show the benefits of using VM-LM to fit additional tasks in an already very tightly-fitted cluster.

Table 6 details those experimental results:

Parameter (average per minute)	Cluster Size (nodes)			
	100% (original)	99%	98%	97%
Nodes Count	12460.39	12332.92	12218.61	12081.30
Tasks Count	132061.15	132057.96	132057.54	132055.86
Global CPU Usage Ratio	43.64%	44.09%	44.52%	45.05%
Global Memory Usage Ratio	62.05%	62.72%	63.39%	64.08%
Idle Nodes	76.41 (0.61%)	53.29 (0.43%)	58.96 (0.48%)	75.87 (0.63%)
STA Nodes	479.91 (3.85%)	480.28 (3.89%)	404.13 (3.31%)	448.67 (3.71%)
TA Nodes	566.20 (4.54%)	572.24 (4.64%)	485.55 (3.97%)	500.75 (4.14%)
PA Nodes	8507.49 (68.28%)	8412.71 (68.21%)	8866.13 (72.56%)	8663.88 (71.66%)
DA Nodes	2818.11 (22.62%)	2800.30 (22.71%)	2361.64 (19.33%)	2339.31 (19.35%)
Overloaded Nodes	12.28 (0.10%)	14.11 (0.11%)	42.20 (0.35%)	62.07 (0.51%)

Table 6. Throughput results (97%-100% cluster)

7.9 Migration Cost

The MASB framework relies on a VM-LM feature to balance workload by moving running tasks across Cloud nodes. While the VM-LM process is reasonably cheap in terms of the computing power, it does incur a non-trivial cost on the Cloud's infrastructure. In order to avoid excessive networks transfers, NAs carefully decide which tasks will be migrated out from a given node. To score candidate tasks, the SCS function is used which takes the task's estimated migration cost into consideration as well as released resources (see 5.1 for more details).

Unexpectedly, when searching for ways to lower the total migration cost, although modifications of SCS function seemed to be the most palpable place to start, significantly better results were not obtained. Based on experience from previous experiments, it was discovered that the biggest reduction in task migrations was achieved by improving the quality of

the initial task allocation. Therefore, further experimentation focused on testing variants and combinations of the score functions.

As previously mentioned, initially MASB implemented a single SAS function which prioritised the scattering of tasks amongst nodes. With introduction of SAL (detailed in subsection 6.1), the SAS function was split into SIAS and SRAS functions biased towards opposite allocation types, namely PA and TA. However, during the study of the impact of frequent re-allocations on the total migration cost, it was found that those scoring functions can be further improved by introducing GAIN variants. Figure 11 presents the evolutions of scoring functions:

Figure 11. Scoring functions evolution

The GAIN variants of SIAS and SRAS functions are defined here as SIAS_GAIN and SRAS_GAIN respectively:

$$SIAS_GAIN = SIAS(T') - SIAS(T)$$
$$SRAS_GAIN = SRAS(T') - SRAS(T)$$

where T is the current set of allocated tasks, and T' is the candidate set of allocated tasks on a given node. Additionally, cases when a node would lower its AS as a result of migrations have a zero score.

In the GAIN variants of scoring functions, the relative AS gains are prioritised over the absolute AS values for an individual node. For example, given the scenario in which the task migration to node A would change its AS from 0.1 to 0.4 (a 300% gain), while the same task could also be migrated to node B, changing its AS from 0.4 to 0.6 (a 50% gain), the former option will be selected as yielding a higher gain (since 300% is greater than 50%) regardless of the potentially higher absolute score value of node B.

The combination of SIAS_GAIN and SRAS functions was most efficient, i.e. the total average migration cost as well as the average cost per task migration were lowest, while ASR remained virtually unchanged. Nonetheless, the good results were also yielded with the combination of SIAS and SRAS. Table 7 presents the results under the variants of the scoring functions:

	Scoring Functions			
Parameter (average per minute)	**SIAS SRAS**	**SIAS SRAS_GAIN**	**SIAS_GAIN SRAS**	**SIAS_GAIN SRAS_GAIN**
Total Migration Cost [GB]	1490.65	7008.30	1252.50	5925.41
Cost per Task Migration [MB]	338.09	795.90	339.02	954.21
Idle Nodes	83.54 (0.67%)	105.80 (0.85%)	76.14 (0.61%)	79.33 (0.64%)
STA Nodes	495.09 (3.97%)	687.77 (5.52%)	490.42 (3.94%)	654.18 (5.25%)
TA Nodes	656.67 (4.54%)	560.65 (4.50%)	558.57 (4.48%)	547.38 (4.39%)
PA Nodes	8515.95 (68.34%)	8492.21 (68.15%)	8511.61 (68.31%)	8451.12 (67.82%)
DA Nodes	2785.23 (22.35%)	2586.43 (20.76%)	2810.95 (22.56%)	2707.73 (21.73%)
Overloaded Nodes	14.88 (0.12%)	27.54 (0.22%)	12.67 (0.10%)	20.61 (0.17%)

Table 7. Results comparison of SAS, SIAS and SRAS (migration cost)

The experiment showed that focusing on the node AS's absolute value as well as value gain are both viable strategies during the initial task allocation (with the former being relatively better). However, it is the selection of the task re-allocation strategy that is crucial and should be dedicated to maximising the absolute value of the node's allocation score. As mentioned previously, the majority of tasks scheduled on the GCD cluster are short-lived batch jobs which tend not to have high resource requirements [44]. As such, there is no need to carefully fit them to a node. As a result of their limited time on the cluster, the chance of re-allocation is low. Long-running services, however, should be fitted tightly onto available nodes and continue to run there due to the additional cost of further re-allocations because of the typically large amounts of used memory.

7.10 Scalability Study

The final step in the experiments was to examine the scalability of the MASB framework. Due to the simulation's high computational requirements, its one-minute time slices were split into 'rounds', in which every NA could both respond to migration requests as well as send its own requests, although sent requests would be unanswered until the next 'round'. This meant that the simulated scenarios were as realistic as possible whilst also emulating massive Cloud installations.

Such a long simulation was necessary in order to achieve reliable and quality results. The month-long GCD workload traces were produced by an actual Cluster system and contain many real-world scenarios which would not be possible to synthesise

in any other way. Special thanks are due to University of Westminster IT staff which provided a massive help and support during those experiments.

Table 8 demonstrates the results achieved through the multiplication (here: two, four and eight times) of the original GCD workload; it also highlights the lack of changes in ASR values:

Parameter (average per minute)	Cluster Size (nodes)			
	12.5k (original)	25k (2x)	50k (4x)	100k (8x)
Nodes Count	12460.70	24921.49	49842.99	99685.97
Tasks Count	132061.35	264155.80	528336.38	1056645.92
Idle Nodes	71.61 (0.57%)	95.82 (0.38%)	226.42 (0.45%)	413.03 (0.41%)
STA Nodes	492.67 (3.95%)	805.60 (3.23%)	1920.99 (3.85%)	3868.22 (3.88%)
TA Nodes	570.37 (4.58%)	962.14 (3.86%)	2232.10 (4.48%)	4300.70 (4.31%)
PA Nodes	8502.06 (68.24%)	18118.11 (72.71%)	34102.21 (68.42%)	68999.49 (69.22%)
DA Nodes	2812.74 (22.57%)	4914.55 (19.72%)	11324.77 (22.72%)	22031.79 (22.10%)
Overloaded Nodes	11.26 (0.09%)	25.25 (0.10%)	36.49 (0.07%)	71.83 (0.07%)

Table 8. Scalability tests – 12.5k, 25k, 50k and 100k nodes

MASB was able to orchestrate a cell size of 100k without a noticeable scalability cost and without crossing the limit of 0.5% overloaded nodes. With the current MASB framework implementation, the simulation of this size took around nine months on a single node of the University of Westminster.

Google has never disclosed the size of their largest cluster, but it has been noted in [51] that Borg computing cells are similarly sized to the clusters managed by Microsoft's Apollo system, which have in excess of 20k nodes [7]. A 12.5k node cells in GCD traces have been described as 'average' or 'median', cells with fewer than 5k nodes have been called 'small' or 'test' [51]. Additionally, [51] gives an example of a larger cell C, which is 150% the size of cell A and therefore also approximately 20k nodes. As such, in this research it is assumed that the computing cell of the large Borg is around 20-25k nodes.

Therefore, as demonstrated, the designed multi-agent load balancing strategy scaled beyond the original GCD workload without incurring noticeable scalability costs. The paradigm of offloading the scheduling logic onto nodes themselves has the following benefits: (i) it enables the implementation of more complex scheduling schemas as the nodes resources can be used for that purpose; (ii) the computing power dedicated to cluster orchestration increases together with the Cluster size (so allowing for greater scalability); and, (iii) limits the amount of communications required to maintain up-to-date Cluster state information. The result of such a schema is the ability to enlarge the computing cells to the sizes of 100k nodes while preserving a good throughput and performance.

8. EXPERIMENTAL RESULTS

During work on the Cloud load balancer prototype, a number of publications were examined and later compared with the proposed MASB design. Aside from the solutions presented in section 2, the following three systems listed in the subsections below have been found to share a degree of similarity with MASB.

8.1 ANGEL System

The ANGEL system [64] is based on a concept wherein a multi-agent system manages its workload in a virtualised Cloud environment. This solution also takes advantage of the VM-LM feature to re-allocate running tasks to an alternative node if necessary. While the basic concept of ANGEL and the MASB system is similar, the design of the architecture and features differ substantially:

(i) Within ANGEL each task is represented by Task Agent created upon task arrival and destroyed when the task is complete. VM Agent represents a VM hypervisor running on a physical node and accepting/rejecting tasks. In comparison, during the development of MASB, it was found that the sheer number of tasks made it impractical to create an entity for each task responsible for its allocation; given this, the responsibility was assigned to NAs. In MASB, NAs themselves are responsible for keeping their node stable and offloading overloading tasks to alternative nodes. Therefore, MASB can potentially support very larger number of tasks. Indeed, during simulations one million tasks were continuously managed.

(ii) In ANGEL, Manager Agent acts as a leader for this computing cell and stores the complete system state in a 'VM Information Board'. VM Agents are constantly updating Manager Agent as to changes in their state, such as available resource (CPU and memory) changes, VM creations and cancellations. The ANGEL system assumes that the stored system state is always current, and Manager Agent this information to match Task Agents with VM Agents. In MASB a subnetwork of BAs has responsibility of caching the global system; however, this information is accepted as outdated by design, and so system uses it only for building initial

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

candidate nodes list which is then sent to NAs. Therefore, MASB doesn't rely on accurate and timed updates from system nodes and the actual task allocation is resolved later between NAs themselves.

(iii) MASB is focused on a Cluster throughput and scalability whereby resource usages gaps are reduced, and tasks are fitted into available nodes. The focus of the project was to achieve tightness of task allocations no worse than in the GCD traces while improving scalability. The aim of ANGEL is to guarantee the ratio of tasks guaranteed to meet their deadlines which are also priority-adjusted. Therefore, ANGEL seems to be more aimed at high churn of short-term tasks, while MASB is designed to support mixed-workload consisting of batch jobs as well as long-lived services.

The authors of ANGEL also tested their solution on GCD traces. In so doing, they acknowledged the difficulty of conducting experiments on the whole month-long traces because of the enormous count of tasks in the trace logs. As such, they performed their experiments exclusively on the 18th day of traces, which has been recognised as being the most representative time period in GCD traces [33]. However, the results presented use different metrics and do not specify further details of the experiments, such as whether authors also matched task constraints and whether tasks were allocated with regards to handing RUS.

8.2 US Patent 5,031,089

[25] filed a patent which described a set of routines that could be deployed on nodes in order to balance system-wide workload. The first routine periodically examines a number of jobs on the node's queue and computes the 'workload value', which is then provided on request to other nodes by the second routine. The third routine, meanwhile, is triggered periodically when the node is idle, and at the end of each job completion. This routine contains the main bulk of load balancing logic and evaluates whether the node's 'workload value' is below a pre-established value that would indicate that the node is relatively idle. If the node is recognised as being under-utilised and available for more jobs, then the routine will poll all the other nodes for their 'workload value', and transfer jobs from the node with the highest 'workload value' to its own queue.

The feasibility of this invention was validated via several simulations, although those results are not shared in the cited patent. The authors list several assumptions made during the performance testing of this study, such as the homogeneity of all the tasks and their resource requirements, as well as the assumption that the job's transfer cost is negligible. The main criticism of this solution is that it oversimplifies the Cluster workload's model, and it omits the continuous changes of resources used by jobs. Only the job's queue length was used as 'workload value'. Furthermore, only non-started jobs can be transferred to alternative nodes. The solution relies on polling all nodes in the cluster for their utilisation levels, which in a large cluster might be not feasible and may create a bottleneck.

8.3 US Patent 8,645,745

[5] notes that there is a problem when a centralised job scheduler needs to pass through a large number of nodes in order to find one which can be used to run the task, and proposed a solution whereby each node is continuously scanning a shared-file to determine which job could be executed on this node. When a job requires multiple nodes, the one on the nodes becomes a primary node, which then assigns and monitors the job execution on the multiple nodes.

In comparison to MASB, the main similarity is that there is no centralised manager to assign tasks to nodes. This means that nodes are themselves responsible for selecting and then running the accepted tasks. However, the main difference is that proposed patented strategy doesn't examine all nodes, and the task is allocated to the first (quickest) scheduler that picks the task. In MASB a task allocation is a multi-step process in which each node tries to increase its AS by selecting the best-matching tasks. Moreover, MASB dynamically manages workload by offloading currently running tasks to the best candidate nodes (with the highest AS score), and, by doing that, the overall system efficiency is increased.

Given that the patent paper provides no results from experiments, it is difficult to directly compare systems' performances.

9. SUMMARY AND CONCLUSIONS

The primary challenge when sequencing a queue of tasks on a cluster is to fit them tightly so as to reduce resource usage gaps. The scheduling algorithm attempts to reduce the situations where a resource on a given node is overly un-utilised at the same time that other resources on that node are mostly allocated. It is extremely important to shrink the gaps in resource utilisation and to allocate them proportionally, especially when initially scheduling new tasks which tend to have balanced resource requirements.

Fitting objects of different volumes into a finite number of containers is known as a 'bin-packing'

problem, and belongs to class of NP-Hard problems. The traditional way of solving NP-Hard problems are metaheuristic algorithms. However, experiments in [49] demonstrated that although metaheuristic algorithms yield good solutions, they do not scale well to the required number of nodes in a Cloud system.

Alternative solutions and a large number of optimisations can be devised, such as caching computed solutions and then retrieving them based on task similarity, multiple concurrent schedulers working on a single data store, and pre-allocating resources for the whole task batches [51]. However, these solutions and optimisations still incur substantial computational costs, and it is inevitable that any model where the head node processes all scheduling logic by itself will eventually work less effectively when the cluster size grows and the frequency of incoming tasks increases.

The MASB framework offers an alternative approach to task allocations in that all the actual processing of scheduling logic is offloaded to nodes themselves. This framework uses loose coupling at every stage of its scheduling flow, meaning that scheduling decisions are made only on locally-cached knowledge and all communication between nodes is kept to minimum. Each node tries to increase its AS by selecting and offloading tasks, with the assumption being that by bettering individual ASs, the global system performance will be improved. This design also takes advantage of the VM-LM feature, where a running program within a VM instance can be migrated on the fly to an alternative node without stopping a program execution.

Design of this schema created a set of new challenges, such as selecting alternative nodes with limited and non-current knowledge about the state of other nodes, estimating the VM-LM cost of migrating a running program, understanding the classifying and scoring functions of the allocation type of a node, and designing the stateless node-to-node communication protocol, to identify just a few.

In this research, realistic (i.e. pre-recorded) workload traces from GCD were used and were run on the AGOCS framework described above as a very detailed simulation. The costs involved were the substantial computing power required to run experiments as well as time, in that a single simulation run took about a month on a forty-core (twenty physical cores + HT siblings) machine. In order to benchmark the research results, original scheduling decisions made by Google's Borg scheduler are examined which are also part of GCD traces. This generated statistics such as total resource usage, the number of idle nodes and production-allocated resources.

When examining GCD traces, it is important to note that Google's engineers did a phenomenal job in first designing and then iteratively improving the Borg system. Incoming tasks are packed very tightly and, although production jobs always have additional resources available to them within defined requirements' limits, the spare resources are efficiently recycled for low priority jobs. Google Cluster has been built upon hardware without direct support for virtualisation, meaning that its orchestrating software design had to accommodate this limitation. This research should be considered an as-if scenario and assumes the availability of the VM-LM feature to shuffle running tasks within a Cluster.

In this research, there was only limited success in terms of improving the throughput of executed tasks on a simulated computing cell. This was mainly due to the constraints arising from handling RUS. During throughput tests, the MASB achieved a similar level as Google's Borg, understood here as the total number of executed tasks. During the progressively more intensive workload, ASR values indicated a degradation in the quality of allocations so that eventually the throughput could be improved by a margin of 2%. However, MASB could achieve higher scalability and run multiple sizes of examined computing cell without noticeable scalability costs. Simulations up to 100k nodes from GCD were tested, yielding relatively comparable results when run with smaller instances of simulations.

9.1 Applications of Technology

In late 2017, a team of marketing experts from IBM estimated that the world generates roughly 2.5 million TBs of data per day, with 90% of all data having been created in the past few years alone [1]. With novel technologies emerging, new devices and sensors being connected, the data growth rate will accelerate even more. To process such vast data streams, new distributed computing models are being designed. In recent years, the trend for software development has been towards Big Data systems and Machine Learning algorithms, specifically:

(i) Big Data systems are characterised by a high degree of parallelism. A typical Big Data system design is based on a distributed file system, where nodes have the dual function of storing data as well

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

as processing it. One program in such a system might need to crunch tens of TBs of data split across thousands of nodes. Even with the ideal allocation of Big Data tasks, where every node is processing data from local storage, a single machine would still need to process GBs of data. In order to speed up this time-consuming process, the partial datasets can be split even further and processed on more nodes;

(ii) Machine Learning is yet another rapidly developing area where there is high demand for computing power. The training algorithms for deep neural networks require multiple iterations over datasets, and the recent research is shifting towards greater parallelism. However, important algorithms such as k-means clustering, alternating least squares, and logistic regression are already very suited to run in parallel. Open Source libraries such as Google's TensorFlow and Spark's MLlib, and the affordability of specialised clusters (e.g. Google's Cloud TPU) makes it easy for businesses as well as researchers to utilise those technologies to enhance their offerings. It can certainly be argued that industries will be adopting Machine Learning in order to increase competitiveness.

Therefore, the organisations which employ those modern technologies are highly likely to build computing cells with even more inter-connected nodes in the near future. To manage larger computing cells, more scalable workload orchestration technologies are required, such as the presented MASB prototype. Experiments have shown that MASB design can run a workload on a large Cloud system (100k nodes) with a throughput comparable to Google's Borg system. It should be noted that larger computing cells are also more economical – Google's Borg demonstrated [51] that running a mixed-workload consisting of short-lived batch jobs and long-running services as well as production and non-production jobs on the same cluster is not only possible, but allows to utilise of available resources more efficiently. Essentially, resource usage gaps are reduced. Therefore, industries such as financials, health or even government, could make monetary savings if their processing centres were joined and more heterogenous workload was introduced in those clusters. MASB is a good candidate for such an integration.

9.2 Future Directions

Although the experimental results prove that it is feasible to deploy the presented decentralised architecture in a live environment, there are several possible other improvements, as listed below:

(i) During experimentations, several nodes remained idle. This effect was a result of iterating only a limited number of nodes while computing a candidate node's set for a given task migration. A potential solution to this issue is a separate size-limited list of relatively under-utilised nodes which would be compulsorily scored each time a BA is issued a GetCandidateNodesRequest request. Such a list could be exchanged separately between BAs;

(ii) The SCS routine (Step 1 in the SAN protocol) is triggered only when the NA detects that its node is overloaded. However, the system could employ a more proactive approach in which the NA would periodically try to offload its tasks in order to improve its AS, even if the node is stable. This would create a secondary mechanism to distribute the load, which would potentially reduce resource utilisation gaps even further. However, this feature would also place additional pressure on BAs and, as such, needs to be carefully balanced;

(iii) In a real-world system it is expected that a number of nodes will experience failure. NA's AI module could maintain a set of blacklisted nodes which repeatedly did not respond to requests. Such a set could be shared with BAs, similar to the way it is implemented in Fuxi [63], and presented to system administrators.

(iv) The MASB prototype does not address fault-tolerance, which is an important aspect of Cloud design. This feature could be realised in multiple ways, such as running cloned instances of tasks, periodically saving process checkpoints, and ensuring the applications' state is synchronised across all its instances. The fault tolerance could also be improved by implementing service/node anti-affinity scheduling strategies where a scheduler tries to allocate replicas of a given service to possibly distanced nodes. In critical failure scenarios, such programs have a greater chance to survive and continue operations. For example, the Kubernetes scheduler implements anti-affinity scoring functions, which gives higher priority to nodes not running services from the same application [24];

(v) Resource usage quotas per user, group or other entity, would make another welcome feature. This is something which is often present in commercial Cluster schedulers. However, it would also require adding an accountancy module with a decentralised dataset in order to maintain scalability. The same mechanism could be used to throttle the submissions of new tasks so as to not extend the Cluster's capabilities;

(vi) The proposed design does not account for task priorities, meaning that tasks are only split into production and non-production groups. Production tasks have committed resources which, under normal circumstances, are guaranteed to be available. However, during critical system-wide failures, such as a power failure or network infrastructure collapse, the system should degrade gracefully (as opposed to an uncontrolled crash). In scenarios where the current workload cannot be sustained, the system should shut down lower priority tasks first and use the remaining available nodes to offload high-priority tasks;

(vii) In this project, it is assumed that NAs and BAs agents are continuously running without breakdowns. Nevertheless, agents are also a piece of software, meaning that they are prone to bugs and errors. As a possible improvement to detect and restore hung agents, a hierarchy model could be introduced in which an agent supervises a number of other agents and restarts them if necessary. This concept is similar to the Akka Actors implementation [41] in which a parent actor manages the failures of its children. Additionally, a hierarchy of BAs could be used to propagate the cluster's state knowledge in a more efficient manner;

(viii) MASB does not attempt to implement locality optimisation when the task's part of a distributed file is processed faster if accessed locally. Currently, GCD task descriptions contain only restrictions which disallow nodes that the task could be executed on. However, adding optional metadata, such as the ID of the distributed file's part, could prioritise a set of nodes and improve the overall cluster performance. This functionality is featured in some of the Big Data frameworks;

(ix) Even though the experimental results presented are of good quality, they suggest a number of potential improvements, especially in locating and then scheduling tasks to idle nodes. One possible improvement could be sharing vector idle nodes between all BAs, and then compulsory prioritising them over utilised nodes;

(x) The Cloud architectures' design is moving towards greater use of Virtual Containers (VC) such as Docker. At the time of writing, Docker does not fully support LM – the integration with Checkpoint/Restore In Userspace tool does not allow the migration of a running application to the alternative container on the fly. Instead, the user must copy checkpoint files and restore them on an alternative node (cold migration). However, the available literature describes early experiments with LM feature [62] and the working prototype was demonstrated in a presentation during the OpenStack Summit 2016 conference in Barcelona (Estes and Murakami, 2016). Once LM becomes the part of mainstream technology, the load balancing strategy presented in this research could be adapted to use VCs;

(xi) MASB estimates the task migration cost, and considers this value when selecting which tasks to migrate out from a node. However, it does not calculate the fact that neighbouring nodes (e.g. those in the same server rack) might offer much faster transfer rates than more remote nodes. Therefore, adjusting the task migration cost by the nodes' distances could improve the overall cluster performance.

The suggested directions of future study and possible expansions as listed above have the potential to improve the results of this research. Nevertheless, the focus of this paper was to research and design a feasible strategy for managing and balancing a workload within a virtualized Cloud system, an objective which has been achieved.

REFERENCES

[1] "10 Key Marketing Trends for 2017 and Ideas for Exceeding Customer Expectations." IBM Marketing Cloud. November 29, 2017. Available from:
https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=WRL12345USEN

[2] Abdul-Rahman, Omar Arif, and Kento Aida. "Towards understanding the usage behavior of Google cloud users: the mice and elephants phenomenon." In Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on, pp. 272-277. IEEE, 2014.

[3] Agnetis, Allesandro, Pitu B. Mirchandani, Dario Pacciarelli, and Andrea Pacifici. "Scheduling problems with two competing agents." Operations research 52, no. 2 (2004): 229-242.

[4] Baker, Kenneth R., and J. Cole Smith. "A multiple-criterion model for machine scheduling." Journal of scheduling 6, no. 1 (2003): 7-16.

[5] Barsness, Eric L., David L. Darrington, Ray L. Lucas, and John M. Santosuosso. "Distributed job scheduling in a multi-nodal environment." U.S. Patent 8,645,745, issued February 4, 2014.

[6] Bigham, John, and Lin Du. "Cooperative negotiation in a multi-agent system for real-time load balancing of a mobile cellular network." In Proceedings of the second international joint conference on Autonomous agents and multiagent systems, pp. 568-575. ACM, 2003.

[7] Boutin, Eric, Jaliya Ekanayake, Wei Lin, Bing Shi, Jingren Zhou, Zhengping Qian, Ming Wu, and Lidong Zhou. "Apollo: Scalable and Coordinated Scheduling for Cloud-Scale Computing." In OSDI, vol. 14, pp. 285-300. 2014.

[8] Brazier, Frances MT, Frank Cornelissen, Rune Gustavsson, Catholijn M. Jonker, Olle Lindeberg, Bianca Polak, and Jan Treur. "A multi-agent system performing one-to-many negotiation for load balancing of electricity use." Electronic Commerce Research and Applications 1, no. 2 (2002): 208-224.

[9] Brenner, Walter, Rüdiger Zarnekow, and Hartmut Wittig. "Intelligent software agents: foundations and applications." Springer Science & Business Media, 2012.

[10] Brooks, Chris, Brian Tierney, and William Johnston. "JAVA agents for distributed system management." LBNL Report (1997).

[11] Buyya, Rajkumar. "High Performance Cluster Computing: Architectures and Systems, Volume I." Prentice Hall, Upper SaddleRiver, NJ, USA 1 (1999): 999.

[12] Cao, Junwei, Daniel P. Spooner, Stephen A. Jarvis, and Graham R. Nudd. "Grid load balancing using intelligent agents." Future generation computer systems 21, no. 1 (2005): 135-149.

[13] Castelfranchi, Cristiano. "Guarantees for autonomy in cognitive agent architecture." In International Workshop on Agent Theories, Architectures, and Languages, pp. 56-70. Springer, Berlin, Heidelberg, 1994.

[14] Chavez, Anthony, Alexandros Moukas, and Pattie Maes. "Challenger: A multi-agent system for distributed resource allocation." In Proceedings of the first international conference on Autonomous agents, pp. 323-331. ACM, 1997.

[15] Gensereth, Michael R., and Steven P. Ketchpel. "Software agents." Communications of the ACM 37, no. 7 (1994): 48.

[16] Gentzsch, Wolfgang. "Sun grid engine: Towards creating a compute power grid." In Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on, pp. 35-36. IEEE, 2001.

[17] Goodwin, Richard. "Formalizing properties of agents." Journal of Logic and Computation 5, no. 6 (1995): 763-781.

[18] Guilfoyle, Christine, and Ellie Warner. "Intelligent agents: The new revolution in software." Ovum, 1994.

[19] Hellerstein, Joseph L., W. Cirne, and J. Wilkes. "Google Cluster Data." Google Research Blog. January 7, 2010.

[20] Hindman, Benjamin, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy H. Katz, Scott Shenker, and Ion Stoica. "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center." In NSDI, vol. 11, no. 2011, pp. 22-22. 2011.

[21] Ilie, Sorin, and Costin Bădică. "Multi-agent approach to distributed ant colony optimization." Science of Computer Programming 78, no. 6 (2013): 762-774.

[22] Jones, James Patton, and Cristy Brickell. "Second evaluation of job queuing/scheduling software: Phase 1 report." Technical Report NAS-97-013, NASA Ames Research Center, 1997.

[23] Kim, Gu Su, Kyoung-in Kim, and Young Ik Eom. "Dynamic load balancing scheme based on Resource reservation for migration of agent in the pure P2P network environment." In International Conference on AI, Simulation, and Planning in High Autonomy Systems, pp. 538-546. Springer, Berlin, Heidelberg, 2004.

[24] Lewis, Ian, and David Oppenheimer. "Advanced Scheduling in Kubernetes". Kubernetes.io. Google, Inc. March 31, 2017. Available from: https://kubernetes.io/blog/2017/03/advanced-scheduling-in-kubernetes

[25] Liu, Howard T., and John A. Silvester. "Dynamic resource allocation scheme for distributed heterogeneous computer systems." U.S. Patent 5,031,089, issued July 9, 1991.

[26] Liu, Peng, and Lixin Tang. "Two-agent scheduling with linear deteriorating jobs on a single machine." In International Computing and Combinatorics Conference, pp. 642-650. Springer Berlin Heidelberg, 2008.

[27] Long, Qingqi, Jie Lin, and Zhixun Sun. "Agent scheduling model for adaptive dynamic load balancing in agent-based distributed simulations." Simulation Modelling Practice and Theory 19, no. 4 (2011): 1021-1034.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[28] Marey, Omar, Jamal Bentahar, Ehsan Khosrowshahi-Asl, Khalid Sultan, and Rachida Dssouli. "Decision making under subjective uncertainty in argumentation-based agent negotiation." Journal of Ambient Intelligence and Humanized Computing 6, no. 3 (2015): 307-323.

[29] Mechalas, John. "Performance Impact of Intel® Secure Key on OpenSSL." Intel Corporation. July 24, 2012. Available from: https://software.intel.com/en-us/articles/performance-impact-of-intel-secure-key-on-openssl

[30] Milano, Michela, and Andrea Roli. "MAGMA: a multiagent architecture for metaheuristics." IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 34, no. 2 (2004): 925-941.

[31] Monteserin, Ariel, J. Andrés Díaz-Pace, Ignacio Gatti, and Silvia Schiaffino. "Agent Negotiation Techniques for Improving Quality-Attribute Architectural Tradeoffs." Advances in Practical Applications of Cyber-Physical Multi-Agent Systems: The PAAMS Collection, vol. 10349 (2017): 183-195.

[32] Montresor, Alberto, Hein Meling, and Ozalp Babaoglu. "Messor: Load-balancing through a swarm of autonomous agents." In AP2PC, vol. 2, pp. 125-137. 2002.

[33] Moreno, Ismael Solis, Peter Garraghan, Paul Townend, and Jie Xu. "An approach for characterizing workloads in google cloud to derive realistic resource utilization models." In Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on, pp. 49-60. IEEE, 2013.

[34] Nguyen, Ngoc Thanh, Maria Ganzha, and Marcin Paprzycki. "A consensus-based multi-agent approach for information retrieval in internet." In International Conference on Computational Science, pp. 208-215. Springer, Berlin, Heidelberg, 2006.

[35] Nong, Q. Q., T. C. E. Cheng, and C. T. Ng. "Two-agent scheduling to minimize the total cost." European Journal of Operational Research 215, no. 1 (2011): 39-44.

[36] Nwana, Hyacinth S. "Software agents: An overview." The knowledge engineering review 11, no. 3 (1996): 205-244.

[37] Othman, Sarah Ben, Hayfa Zgaya, Mariagrazia Dotoli, and Slim Hammadi. "An agent-based Decision Support System for resources' scheduling in Emergency Supply Chains." Control Engineering Practice 59 (2017): 27-43.

[38] Perez-Gonzalez, Paz, and Jose M. Framinan. "A common framework and taxonomy for multicriteria scheduling problems with interfering and competing jobs: Multi-agent scheduling problems." European Journal of Operational Research 235, no. 1 (2014): 1-16.

[39] Isard, Michael, Vijayan Prabhakaran, Jon Currey, Udi Wieder, Kunal Talwar, and Andrew Goldberg. "Quincy: fair scheduling for distributed computing clusters." In Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, pp. 261-276. ACM, 2009.

[40] Reddy, Reddivari Himadeep, Sri Krishna Kumar, Kiran Jude Fernandes, and Manoj Kumar Tiwari. "A Multi-Agent System based simulation approach for planning procurement operations and scheduling with multiple cross-docks." Computers & Industrial Engineering 107 (2017): 289-300.

[41] Roestenburg, Raymond, Rob Bakker, and Rob Williams. "Akka in action." Manning Publications Co., 2015.

[42] Sargent, P. "Back to school for a brand new ABC." The Guardian (March, 12), no. 3 (1992): 12-28.

[43] Schaerf, Andrea, Yoav Shoham, and Moshe Tennenholtz. "Adaptive load balancing: A study in multi-agent learning." Journal of artificial intelligence research 2 (1995): 475-500.

[44] Schwarzkopf, Malte, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. "Omega: flexible, scalable schedulers for large compute clusters." In Proceedings of the 8th ACM European Conference on Computer Systems, pp. 351-364. ACM, 2013.

[45] Sharma, Bikash, Victor Chudnovsky, Joseph L. Hellerstein, Rasekh Rifaat, and Chita R. Das. "Modeling and synthesizing task placement constraints in Google compute clusters." In Proceedings of the 2nd ACM Symposium on Cloud Computing, p. 3. 2011.

[46] Shi, Dongcai, Jianwei Yin, Wenyu Zhang, Jinxiang Dong, and Dandan Xiong. "A distributed collaborative design framework for multidisciplinary design optimization." In International Conference on Computer Supported Cooperative Work in Design, pp. 294-303. Springer, Berlin, Heidelberg, 2005.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[47] Sliwko, Leszek, and Vladimir Getov. "Transfer Cost of Virtual Machine Live Migration in Cloud Systems." University of Westminster. Technical Report. November, 2017: 1-21.

[48] Sliwko, Leszek, and Vladimir Getov. "AGOCS – Accurate Google Cloud Simulator Framework." In Scalable Computing and Communications Congress, 2016 Intl IEEE Conferences, pp. 550-558. IEEE, 2016.

[49] Sliwko, Leszek, and Vladimir Getov. "A Meta-Heuristic Load Balancer for Cloud Computing Systems." In Computer Software and Applications Conference, 2015 IEEE 39th Annual, vol. 3, pp. 121-126. IEEE, 2015.

[50] Tuong, N. Huynh, Ameur Soukhal, and J-C. Billaut. "Single-machine multi-agent scheduling problems with a global objective function." Journal of Scheduling 15, no. 3 (2012): 311-321.

[51] Verma, Abhishek, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. "Large-scale cluster management at Google with Borg." In Proceedings of the Tenth European Conference on Computer Systems, p. 18. ACM, 2015.

[52] Wang, Cheng, Qianlin Liang, and Bhuvan Urgaonkar. "An empirical analysis of amazon ec2 spot instance features affecting cost-effective resource procurement." ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS) 3, no. 2 (2018): 6.

[53] Wang, Gong, T. N. Wong, and Xiaohuan Wang. "A hybrid multi-agent negotiation protocol supporting agent mobility in virtual enterprises." Information Sciences 282 (2014): 1-14.

[54] Weiss, Gerhard. "Multiagent Systems. 2nd edition." MIT Press. 2013

[55] Wilkes, John. "Cluster Management at Google with Borg." GOTO Berlin 2016. November 15, 2016.

[56] Wooldridge, Michael, and Nicholas R. Jennings. "Intelligent agents: Theory and practice." The knowledge engineering review 10, no. 2 (1995): 115-152.

[57] Wyai, Loh Chee, Cheah WaiShiang, and Marlene Valerie AiSiok Lu. "Agent Negotiation Patterns for Multi Agent Negotiation System." Advanced Science Letters 24, no. 2 (2018): 1464-1469.

[58] Xydas, Erotokritos, Charalampos Marmaras, and Liana M. Cipcigan. "A multi-agent based scheduling algorithm for adaptive electric vehicles charging." Applied energy 177 (2016): 354-365.

[59] Yang, Yongjian, Yajun Chen, Xiaodong Cao, and Jiubin Ju. "Load balancing using mobile agent and a novel algorithm for updating load information partially." In Lecture Notes in Computer Science 3619, pp. 1243-1252. 2005.

[60] Yin, Yunqiang, Shuenn-Ren Cheng, T. C. E. Cheng, Wen-Hung Wu, and Chin-Chia Wu. "Two-agent single-machine scheduling with release times and deadlines." International Journal of Shipping and Transport Logistics 5, no. 1 (2013): 75-94.

[61] Yoo, Andy B., Morris A. Jette, and Mark Grondona. "Slurm: Simple linux utility for resource management." In Workshop on Job Scheduling Strategies for Parallel Processing, pp. 44-60. Springer, Berlin, Heidelberg, 2003.

[62] Yu, Chenying, and Fei Huan. "Live migration of docker containers through logging and replay." In Advances in Computer Science Research, 3rd International Conference on Mechatronics and Industrial Informatics, pp. 623-626. Atlantis Press, 2015.

[63] Zhang, Zhuo, Chao Li, Yangyu Tao, Renyu Yang, Hong Tang, and Jie Xu. "Fuxi: a fault-tolerant resource management and job scheduling system at internet scale." Proceedings of the VLDB Endowment 7, no. 13 (2014): 1393-1404.

[64] Zhu, Xiaomin, Chao Chen, Laurence T. Yang, and Yang Xiang. "ANGEL: Agent-based scheduling for real-time tasks in virtualized clouds." IEEE Transactions on Computers 64, no. 12 (2015): 3389-3403.