A Requirement-Driven Mechanism for the Management of Distributed Infrastructures

Souheil Khaddaj and Bippin Makoond

Abstract—The emergence of new service oriented distributed models has raised a number of challenges particularly in relation to the management of distributed infrastructures in dynamic environments, such as the Cloud with changing availability of resources, services and quality of services. In such an environment it is very important that users and applications have some level of assurance that their requirements can be satisfied while trying to optimize the usage of the available resources. This paper presents a new requirement-driven decision making mechanism that is based on a quality assured load balancer for distributed computing systems. We evaluate the approach and demonstrate how it can adapt to user requirements and to the capacity of available resources.

Index Terms—Analytical hierarchy process, distributed systems, cloud computing, load balancers, quality of service (QoS)

1 Introduction

The advancement of computing and information technology has been driven not only by the continuous improvement in hardware infrastructures but also by the development of new operating environments. Indeed over a relatively short period of time, there has been a shift from monolithic infrastructures and applications, to distributed services mainly in the form of Cloud Computing [1], [2], [3] and Service Oriented Architecture (SOA) [4], [5]. However, the shift toward a real utility computing model requires meeting not only users' functional requirements but also their non-functional requirements in the form of Quality of Service (QoS), which is becoming essential with the move toward service orientation.

The universal acceptance of the service oriented approach and of the continuous decentralisation and distribution of software, hardware, and human resources depends on a number of fundamental factors including the ability to provide desired QoS on resources assembled dynamically from enterprises, service providers and customer systems. Quality of Service, which in this context refers to key quality factors and attributes of system infrastructure [6], [7], [8], is the ability of an application to have some level of assurance that user requirements can be satisfied. It can be seen in the form of Service Level Agreement (SLA) between clients and suppliers to provide a service at a specified cost and within a guaranteed time frame [9], [10]. Moreover, the development of service-oriented infrastructures, such as those supporting the Cloud Computing model, require efficient management services with adaptable load distribution capabilities that are driven by user demand.

- *S. Khaddaj is with the School of Computer Science and Mathematics, Kingston University, Kingston upon Thames KT1 2EE, United Kingdom. E-mail: s.khaddaj@kingston.ac.uk.*
- *B. Makoond is with Cognizant Technology Solutions Haymarket House, 28-29 Haymarket, London SW1Y 4SP, United Kingdom. E-mail: bippin.makoond@cognizant.com.*

Manuscript received 31 Oct. 2015; revised 16 July 2016; accepted 24 July 2016. Date of publication 8 Sept. 2016; date of current version 6 Feb. 2019. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TSC.2016.2598338

Although there have been several attempts at designing distributed management systems with QoS driven scheduling algorithms [11], [12], [13], [14], [15], the majority suffer from the fact that they are mainly system-centric and do not necessarily meet the wide range of user requirements and their expectations to obtain a high quality of service, i.e., they are not user-centric, indeed many distributed systems historically have weighed heavily on one requirement namely performance and more recently on availability. Even when decisions and scheduling were based on multiple requirements (multi-criteria), the constructed utility functions were optimised to maximise the overall utility of distributed systems [16], [17]. Moreover, most management systems do not support formal dynamic validation of meeting user requirements, neither they are able to predict the expected system behaviours under a specific working environment. Thus, major challenges remain in the organisation, management and optimisation of distributed infrastructures on the supplier side (system-centric) while meeting the user's many and diverse requirements (user-centric).

This paper proposes a new multi-criteria decision making mechanism, called the BipRyt algorithm, for the management, distribution, control and optimisation of systems resources within distributed systems and which enforces an assured QoS. In fact, it is a combination of two brokering components; a system centric broker that optimises the supplier's resources and a user-centric broker that ensures meeting the user requirements and implementing SLAs. It is an empowerment strategy that provides autonomy to several parts of a system and its novelties lie in the ability to use a multitude of quality attributes for decision making, a set of quality guidelines from the user and applying a reinforcement model to validate these guidelines, when required.

We start by discussing distributed resource management and user requirements. Then, the core components of the BipRyt algorithm are presented with a focus on resource allocation and load balancing that are driven by user requirements. The results of a number of experiments, comparing the BipRyt algorithm with other load balancing strategies, are analysed. We conclude with some suggestions for future work.

2 Resource Management and User Requirements

In order to meet service level agreement demands the management of system resources requires built-in decision making mechanisms that not only ensure that user requirements are met but also that the goals of service providers and the optimisation of resource usage are realised. However, since resource management encompasses a wide range of different scenarios, the decision making for SLAs is a complex procedure, which is particularly true in Cloud environments. Attempts at the simplification of decision-making meant that many SLAs guarantee some QoS typically availability but not necessarily others such as response time.

2.1 Resource Management and Load Balancing

Decision making mechanisms for resource management and load balancing algorithms have been intensively studied since the early days of parallel and distributed systems, with a large number of papers, surveys and books have been published in the literature [18], [19], [20], [21], [22], [23]. Recently, a number of these algorithms have been adopted in virtual infrastructures and Cloud Computing [24], [25], [26], [27]. Such an extensive research has been driven by the criticality of the load balancing issue, the vast number of applications and their diversity, the continuous evolution of the distributed architectures with their wide variety and heterogeneousity, and the developed and commonly used programming paradigms. The application domains range from science and engineering to enterprise computing, e.g., from scientific simulation [28] to semantic services [29]. The infrastructures range from Cluster to Grid and Cloud Computing [30], and the programming paradigms range from procedural and object oriented to parallel and service oriented paradigms [31], [32]. Thus, fundamentally the choice, and suitability, of load balancing algorithms has been driven by the applications and underlying architectures. Consequently, some of the developed algorithms were more suitable for early parallel machines and clusters while others are more suitable for Grids and Clouds.

Over the years there have been many different classifications of load balancing algorithms, which very broadly can be categorised as static or dynamic [22], [33], [34], [36]. In static load balancing, all information regarding all resources and tasks is known in advance thus workload distribution is based on the knowledge of the system [37], [38]. In Dynamic load balancing tasks are allocated to resources as they arrive i.e. dynamically [39], [40]. More specific load balancing strategies have also been studied including, centralized or distributed, local or global, cooperative and non-cooperative, approximate and heuristic, hierarchical etc. [35], [41], [42], and many theories and techniques were used such as game theory, genetic algorithms and fuzzy logic [43], [44], [45]. There are many advantages and disadvantages in each of the above strategies, thus adopting any of the algorithms should very much depend on applications requirements and underlying infrastructure.

However, many decision-making strategies are based on a single requirement i.e., one quality attribute, regardless of its impact on other attributes, for instance: response time, least connections, Round Robin etc. Recently, mechanisms for dealing with energy efficiency requirements have received a lot of attention [46], with major constraints on cost and reliability. In addition a number of works on web services have attempted to deal with QoS aspects [47], [48], [49] but they have focused on the SOA paradigm. Other service computing approaches have focused on multi-criteria requirement capture but not much on resource management [50], [51]. However, with the emergence of Cloud Computing, as a new service driven distributed computing paradigm, QoS aspects and SLA have become essential. These include the use of MAPE (Monitoring, Analysis, Planning, Execution) loop for the management of cloud infrastructures [57]. Although the majority of works have focused on the Infrastructure as a Service (IaaS) layer some QoS aspects have also been considered at other layers such Platform as a Service (PaaS) [62], [63]. But, many challenges remain as PaaS interacts with both IaaS and SaaS (Software as a Service) with considerable focus on the QoS of the infrastructure [64]. Overall, most QoS works have concentrated on a limited number of quality attributes mainly availability and performability and many still rely on the Round Robin load balancer to manage their workload which does not offer an optimal solution [58], [59], [60], [61]. In summary, many of the approaches only partially meet user requirements, and those that attempted to meet multiple requirements were mainly focused on maximising global system utilisations while meeting the minimum needs of user applications and requirements [16], [17].

2.2 User Requirements Capture

User requirements particularly in terms of QoS have changed from the early days of distributed computing when performance, and perhaps scalability, were the most important factors, and now includes a wider range of quality factors such as reliability, availability, usability etc. Thus, QoS-based scheduling became very important particularly in enterprise applications. However, the analysis of diverse user requirements is a multi-criteria problem and require a multi-criteria decision making approach. In this work we adopt the Analytical Hierarchy Process, which seems to have replaced other approaches such as Multi Attribute Utility Theory due to its theoretical soundness [53], [55]. By using AHP, the requirements engineer can also confirm the consistency and reliability of the result and prevent subjective judgment errors.

In summary AHP is used in decision making, 1) to elicit preferences for certain objectives comparatively to other objectives and 2) to give the best (or several best), solution (s) from a range of potential solutions. As shown in Fig. 1, there are a number of steps in the AHP process

1. Define the problem with its main objectives.
2. Lay out the elements of the problem as hierarchy.
3. Establish element comparison within matrices.
4. Calculate element priorities and consistency check.
5. Calculate the priorities and produce a priority vector.

The AHP represents a weight matrix that maps attributes against attributes using the Saaty scale of prioritization (Table 1) for assigning the values of importance to each attribute. AHP also provides a Consistency Index (CI) and Consistency Ratio (CR), based on the maximum eigenvalue (Perron root) of the matrix $\lambda\max$, to validate the consistency of the AHP results [53], [54].

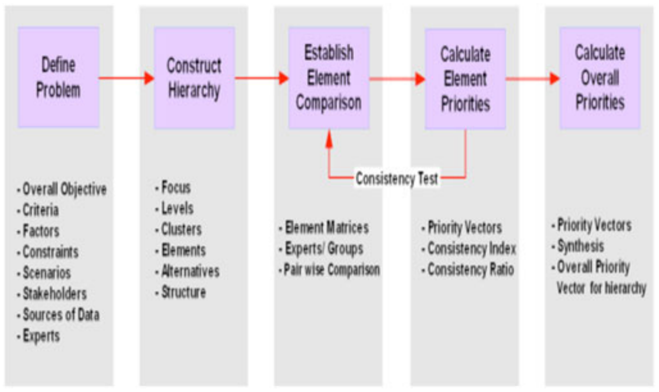


Fig. 1. AHP process flow.

Moreover, in order to ensure that the values, which are assigned to attributes, represent robust and accurate analytical values, a combination of CI and CR indexes are factored into the process to ensure that the achieved results are within an acceptable range of values. However, the range depends on the number of attributes that are compared (Table 2).

Using AHP analysis we are proposing a new multi-criteria mechanism, namely the BipRyt algorithm, that takes into account many quality attributes while preserving the overall quality of the system by continuously assessing the impact of the attributes against each other. The mechanism establishes its decisions by combining quality attributes. It is implemented on a new concept called run time quality assurance which ensures that quality attributes are preserved at any time and any cost.

3 The Mechanics of the BipRyt Algorithm

The BipRyt algorithm is a decision-making mechanism based on the availability of computational resources, associated rules of usage, and defined rules for a specific user, group of users or the system as a whole. It is based the price-driven model [56] and it allows for the optimisation of resources during the system life cycle. This is achieved through a rule-based system where the rules can be local or global. Part of the mechanism's responsibility is to manage conflicts among the rules, focusing on the problems of racing condition and resource starvation, and hence providing a balance between the two axes. There are five basic principles in this mechanism

1. All software components or software agents consumes resources during execution.
2. Each software agent has several resources such as CPU, memory, and bandwidth. Each resource is associated with a quality attribute. For example, if the resource is processor, the associated attribute is CPU time (processing time), which is measured in ticks or seconds while an attribute such as memory is measured in gigabytes.
3. A quality attribute of a resource can be assigned a numerical value called energy level representing the level of a resource consumption that are measured at periodic intervals, for example 50 percent of CPU time or 75 percent of memory size.
4. The health of the system depends on the energy levels, which have to be measured and controlled. A healthy system is attained by avoiding the starvation or overloading of resources, consumption of which can be measured using energy levels.
5. The energy levels decrease as the resources are consumed and increase when resources are released.

TABLE 1
The Saaty Rating Scale

Level of Importance	Description
1	Equal importance
3	Somewhat more important
5	Much more important
7	Very much more important
9	Absolutely more important.
2, 4, 6, 8	Intermediate values

TABLE 2
Consistency Index Matrix

No of Attributes	1	2	3	4	5	6	7	8
CI	0.0	0.0	0.58	0.90	1.12	1.24	1.32	1.41

We now introduce two core modules of BipRyt, namely the Perceiver and Decider. The Perceiver is the part of the software agent that collects snapshots of information on the energy levels of each resource that are consumed at periodic intervals. The Perceiver, hence, builds up a history of energy levels for each Quality of Service, which is then fed to the Decider. The Decider performs some statistical analysis over the recorded quality attributes and cross references the quality model with the QoS priority. Depending on the results, the Decider decides which agent is the healthiest to handle or process more information in the system.

3.1 Perceiver

Within the system the software agents receive data to be processed, which consume a defined number of resources. Each software agent has a list of resources and each resource represents a quality attribute, which has its own container. The quality attributes are represented in terms of energy levels, which are held in the quality containers. These containers have numerical values, which determine how much energy every agent has for a particular quality attribute. The list of resources is the same for all agents (CPU, memory etc.) but the resources' capacities can be different for different agents, for example different memory size. Moreover, all agents process the same quality attributes as derived from user requirements, but depending on the available resources the energy levels, for the same attribute, on different agents might be different.

In addition, the containers do not only link the energy level to a numerical value but also create a number of energy level areas defining the risk values of meeting, or not meeting, quality requirements. In fact a quality container is partitioned in three distinct areas. There is the Low Risk Area (LRA), Medium Risk Area (MRA) and the High Risk Area (HRA). HRA means that the value of a particular quality attribute has dropped to a level that constitutes a high risk for the system. For example, if a CPU hits say 90 percent usage, this indicates HRA, which is fed to the Decider to avoid node overloading. On the other hand LRA means that the value of a given quality attribute has reached a level that

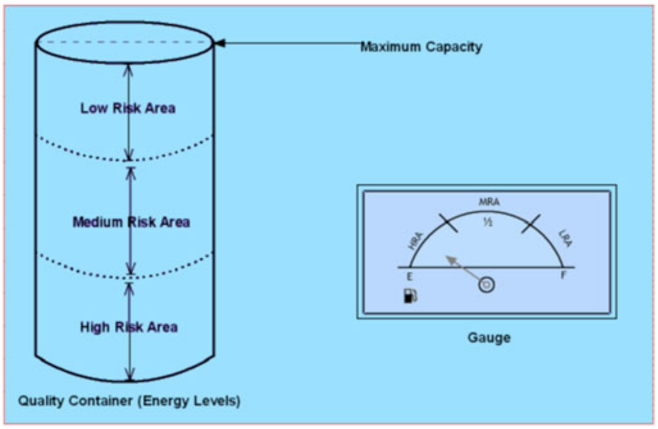


Fig. 2. Illustrative model of the perceiver within each agent.

shows a low risk, which is fed to the decider to avoid node starvation. The objective is to meet user requirements by avoiding the HRA region through intelligent management and distribution of workload across the quality attributes. It is important to note that all agents follow the same model as shown in Fig. 2.

At system run time the energy level rises and drops along the three risk areas. The Perceiver has the ability to monitor the energy levels of the quality container, which is represented through a gauge system, connected to each quality attribute. When energy is consumed, the indicator on the gauge moves towards the HRA (left). However, as agents complete the processing, the energy level increases and added to the container. This will result to the indicator moving towards the LRA (right). The Perceiver will record the indicator reading per processed data. When the Decider requests the information, the Perceiver sends a history of the indicator values, which builds up a feedback system based on the usage of quality attributes.

3.2 Decider

The decider is the decision-making module of the BipRyt algorithm, which is at the core of the system's resource management. The success of the decision-making is based on a number of basic requirements, including not only the provision of dynamic and efficient services and resources but also the enforcement of a certain level of QoS to the users. This responds to the increased QoS provision requirements, particularly for enterprise and Cloud applications where there are higher expectations of users to receive high quality services at an agreed price and agreed time scale.

In order to meet these requirements suitable user requirements components, for QoS evaluating, matching and enforcing, and scheduling and load balancing components, for the management of system resources, are needed. This can be achieved by a combination of two brokering modules; a user-centric broker that ensures meeting the user requirements and a system centric broker that manages and optimises system resources (Fig. 3).

The user centric broker defines the QoS mapping strategy of requirements from user to resources and continuously communicates with the system centric broker. The QoS mapping defines the minimum capacities or resources, which are needed to meet user requirements. Examples are

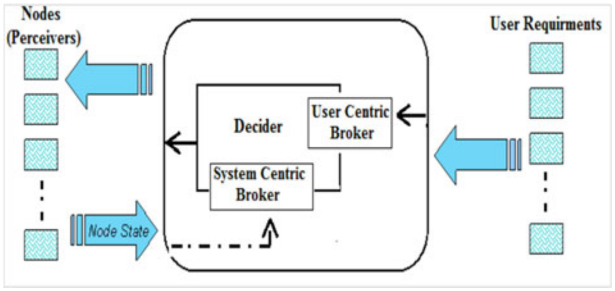


Fig. 3. Illustrative model of the decider.

minimum CPU and memory needs, reliability, type of traffic, throughput etc. The QoS mapping can also take into account other user requirements such as user's budget and resource prices.

The other part of the decider is the system centric broker with permission to access directly the raw resources such as CPU, memory, and bandwidth through the local management system. It should also regularly communicate with the Perceivers to get updates on their status, the available resources and level of usage of individual nodes. The Decider is able to perform some analytical calculations over the data, gathered from the Perceivers, upon which decisions are made which are taken into account when allocation resources to meet user requirements.

3.3 The Mechanism

At the initial stage the BipRyt algorithm needs to be aware of its current operating environment and system configuration, which in this context consists of the nodes with parameters such as the number of CPUs, memory size and bandwidth, representing the system configuration variables. Thus, it starts by recognising and discovering its current system configuration and the status of its nodes; a process which it repeats continuously thereafter by gathering data from the Perceivers as shown in Fig. 4. When receiving the list of energy levels from the Perceivers, the Decider builds an Energy Matrix (EM) of agents by quality and populates the array with values from the list. The Matrix is built to reflect both system resources and QoS

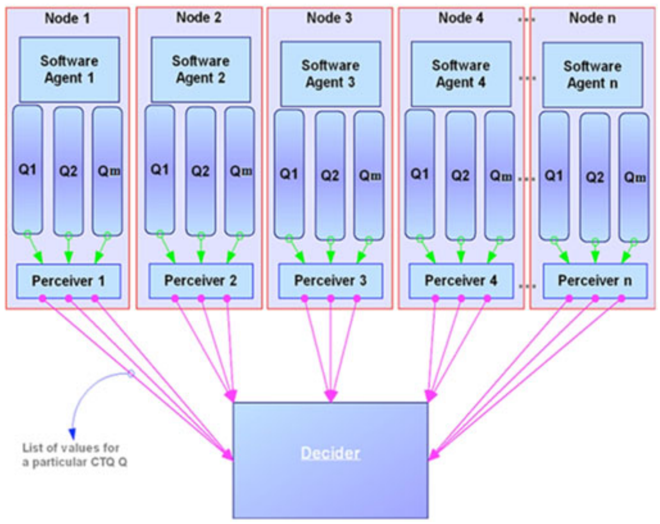


Fig. 4. BipRyt data gathering process.

requirements. The Matrix is then normalised followed by the application of AHP.

As mentioned earlier AHP has been extensively used in the domain of decision-making and the density of its application is at the requirement and design phases of many engineering projects including the software development life cycle. Our novelty lies in the fact that we have shifted the capability of the AHP from being a subjective tool to embedding it into the dynamics of the system to make decision at runtime. This is a move from AHP being a planning tool to being a run time quality assurance tool.

Thus, the process of evaluating quality attributes is automated, which involve the use of qualitative and quantitative tools to help the Decider to evaluate non-functional requirements, particularly the ones that are Critical to Quality (CTQ). In order to make the algorithm aware of a given quality model, the AHP process is integrated within the mechanism, hence within the program. AHP provides a set of instructions that are automated through a sequence of actions, which includes building the quality model, analyzing the model, prioritizing the quality attributes using their AHP weights, calculating Consistency Index (CI) and Consistency Ratio (CR) using the maximum eigenvalue (Perron root) of the matrix λmax. Human input is reduced to choosing a number (1 - 9) from Table 1 representing the importance of a particular QoS, which is either extracted (parsed) from an SLA template or entered directly by the user.

By transferring the responsibility of evaluating the quality attributes from a design time to run time, the decision-making has been pushed forward through the development life cycle, into the system at deployment phases, which result in reducing the modelling uncertainty as more data became available. The proposed algorithm embeds the concept within its decision-making matrix. In doing so the BipRyt algorithm follows a number of steps, which are triggered when the Decider receives the list of energy levels per quality attribute per agent. First, the mean of energy level per quality attribute for each agent is calculated by the Decider. Then, the Decider builds a mean Energy Matrix of agents by quality:

$$EM = \begin{bmatrix} e(a_1q_1) & e(a_1q_2) & e(a_1q_3) & \dots & e(a_1q_m) \\ e(a_2q_1) & e(a_2q_2) & e(a_2q_3) & \dots & e(a_2q_m) \\ e(a_3q_1) & e(a_3q_2) & e(a_3q_2) & \dots & e(a_3q_m) \\ \dots & \dots & \dots & \dots & \dots \\ e(a_nq_1) & e(a_nq_2) & e(a_nq_3) & \dots & e(a_nq_m) \end{bmatrix},$$

where e is the energy level per quality attribute q per agent a, n is the number of agents and m is the number of attributes per agent. The matrix is then normalised by order of the quality attributes into the Normalised Energy Matrix (*NE*).

$$NE = \begin{bmatrix} ne(a_1q_1) & ne(a_1q_2) & ne(a_1q_3) & \dots & ne(a_1q_m) \\ ne(a_2q_1) & ne(a_2q_2) & ne(a_2q_3) & \dots & ne(a_2q_m) \\ ne(a_3q_1) & ne(a_3q_2) & ne(a_3q_2) & \dots & ne(a_3q_m) \\ \dots & \dots & \dots & \dots & \dots \\ ne(a_nq_1) & ne(a_nq_2) & ne(a_nq_3) & \dots & ne(a_nq_m) \end{bmatrix},$$

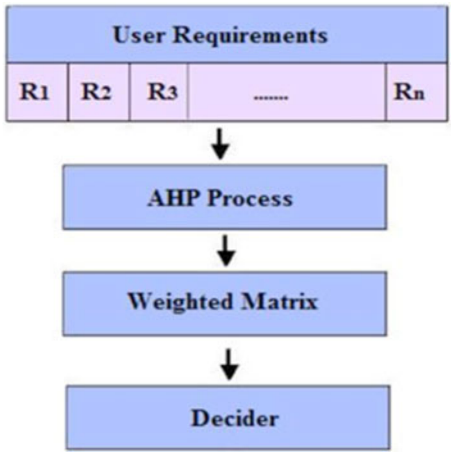


Fig. 5. BipRyt user requirements AHP process.

where:

$$ne(a_iq_j) = e(a_iq_j) \times \frac{1}{\sum_{j=1}^{m} e(a_iq_j)}$$

for each quality attribute q_j.

Finally, in order to assess which quality is most important for the system the Decider consults a prioritization table, which is fabricated by Analytical Hierarchy Process.

Thus, the users configure the AHP by prioritizing their requirements (*R*) and quality attributes (according to their point of views). The input table is then processed to give the weights of quality attributes (*W*) under AHP (Fig. 5). The outcome shows the relative importance of a quality attribute compare to the others at assessing time.

$$W = \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_m \end{bmatrix}.$$

Hence, within the BipRyt operations, each normalised value of each quality for the agents is multiplied by the corresponding quality value. The new values of quality attributes are added together for each agent. The sum shows the distribution load of an agent.

$$\sum_{j=1}^{m} (ne(a_iq_j) \times w_j).$$

Due to the fact that the distribution is calculated, based on the priorities of the users and on the energy levels BipRyt maximizes the opportunity for a system to conform to the user's desires or needs in terms of quality requirements.

It is important to note that AHP plays a core part in the BipRyt algorithm. In fact, there are many advantages in integrating AHP within the algorithm, many of which have been discussed earlier. In summary, AHP is used for capturing user requirements, particularly in terms of a multitude of quality attributes, building a quality model, analyzing the model and prioritizing the quality attributes, thus helping in making decisions for the welfare of the system. In addition, AHP instructions are automated and integrated into the Decider's code base. Thus, the whole process from

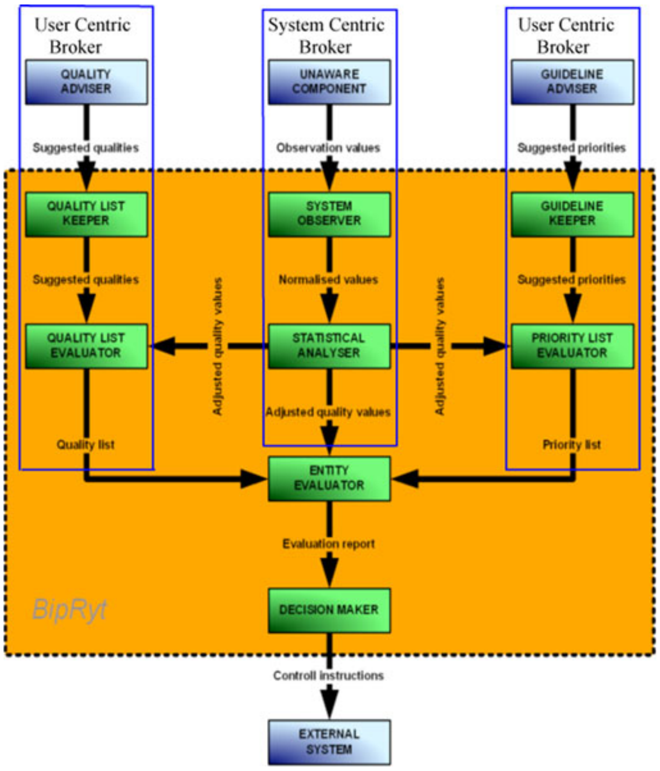


Fig. 6. Architectural model of the BipRyt system.

prioritizing quality requirements to resource allocation can be automated and optimized.

3.4 Deployment

Having presented the different aspects of the decision-making mechanism we move on to the deployment strategy. As it can be seen in Fig. 4 a traditional centralised solution has been adopted at this point mainly because it exhibits good decision making since it is capable of 'seeing' the global view of the entire system. Achieving a good decision-making capability in terms of resource management, while meeting a multitude of user requirements and QoS, is the primary aim of this work. Therefore, the comparison is made with other load balancing algorithms also in a centralised mode.

Once the suitability of the proposed algorithm and its benefits in terms of decision-making is established, it will be extended into a distributed strategy with multiple BipRyt deciders. This will address the lack of scalability and fault tolerance of the centralised approach. Still, with the low cost of available hardware it is relatively straight forward to implement hardware redundancy for the decider i.e., redundant decider in either active/passive mode or active/active mode which can improve the reliability of the system.

But, with distributed decision making and multiple deciders, a cooperation strategy will have to be identified i.e., are the deciders working cooperatively or non-cooperatively. In a non-cooperative environment, individual deciders act autonomously and make decisions regarding their own user requirements and their own local resources independently from other deciders and without consideration of the decisions impact on the whole of system. In the cooperatives case, each decider is responsible for its own part of the resources and requirements, and is concerned with making decisions in concert with the other deciders in order to meet global system requirements.

4 The Architecture Model

In this section we explain the generic architectural model that employs the BipRyt algorithm. BipRyt will manage the behaviour of the system entities (for instance, the software agents) and provides a generic decision making mechanism. Signals (energy levels) from the software agents (Perceivers) will be recorded by the Decider and statistically evaluated. Fig. 6 illustrates the generic, abstract architecture and its various components.

The architecture provides a detailed view of the system which reflects the high level view shown in Fig. 3, i.e., the details of the functionality of the inner working of the Decider as well as the external, to the Decider, components. At the top level of the architecture are the external components that feed information to the Decider which includes 1) The Guideline Adviser, representing the priority values per quality attributes (w), and Quality Adviser representing the initial suggested qualities (q) with both advisers representing user requirements in terms of quality and priority that feed into the user centric broker of the Decider, and 2) Unaware Component representing the Perceivers feeding into the system centric broker of the Decider. Internally, within the decider, there are a number of different components some of which are dealing with quality configuration and processing user requirements while others such as System Observer are processing the regular feed from the Perceivers. Finally, at the low level of the architecture is the External System, which receives the directives in the form of Control Instructions, from the Decision Maker, which can trigger a change in the external system behaviour.

4.1 Processing User Requirements

At the top level the Guideline Adviser feeds the suggested priority distribution values to the algorithm. The suggested priorities are represented by a list of directives from the Guideline Adviser reflecting its wishes and vision on how the algorithm should perform and what quality attributes need to be preserved. This in essence represents the priority values per quality attribute as defined by the user requirements in the form of a table.

Internally, the Guideline Keeper is responsible for keeping and maintaining the directives that were received from the Guideline Adviser. Directives are fed to the Keeper in the form of Suggested Priorities, which are provided to any requesting part of the system through the Priority Evaluator (PE). The purpose of the PE is to query Suggested Priorities from the Guideline Keeper and to evaluate them for correctness, completeness and validity. The other function of the PE is to modify some or all items in the Suggested Priorities list according to the decisions made. After any potential correction, PE produces an ultimate Priority list that is the resulting set of priorities used to make final advices/decisions. It is also PE's responsibility to respond and supply the evaluation results to any requesting part of the system.

Quality Adviser, the second external entity, has the responsibility of feeding the system with the initial suggested qualities, which represent the Quality Advisers vision of the system. Each item supplied by the Quality Adviser may or may not be accompanied by Suggested Priorities of the Guideline Advisors.

Internally, the Quality List Keeper is responsible for keeping and maintaining Suggested Qualities list supplied by the Quality Adviser. It is also the responsibility of the Quality List Keeper to supply its list of Suggested Qualities to any requesting part of the system through the Quality List Evaluator (QLE). The purpose of the QLE is to query the Suggested Quality list and to evaluate them for correctness, completeness and validity. The QLE can modify some or all items in the Suggested Quality list according to its own observed values, respond and supply the evaluation results to any requesting part of the system. Ultimate Quality Attributes are the resulting set of qualities that the system uses to make its final advice, derived from the Suggested Qualities.

In summary, user requirements are obtained through AHP analysis, which is an integral part of the user centric broker, as tasks arrive. As the identified qualities and priorities are fed through the system, an implicit voting [52], is deployed by the Quality List Evaluator, Priority List Evaluator in consultation with the Statistical Analyzer. Thus, tasks receive different priority levels in the allocation/scheduling processes with the aim of meeting both user requirements and efficient utilization of the system. Like other resource allocation decision-making mechanisms, tasks are submitted through a queueing system. The resource allocation policy is fundamentally based on the weighted and ranked quality attributes associated with the submitted tasks, and their priority values, and the ranked hardware resources (please see Section 4.3 for further discussion).

4.2 Processing System Resources

At the top level is the Unaware Component, an external entity, which belongs to a system that the algorithm is making decisions for. Unaware Component responds to requests regarding its current state in terms of energy levels of quality attributes. Unaware components produce Observation Value list of qualities and report their usage to the system. Each item in the Observation Value list may or may not be accompanied by an entry in the Suggested Qualities provided by Quality Adviser.

Internally, the System Observer is the part of the system with the purpose of accepting and requesting Observed Values from one or more Unaware Components. System Observer responds with its current list of items to any request from internal components, and performs normalisation of data gathered which allows operations on balanced data, thus any further comparisons are performed on equally scaled values. Then, the Normal Quality Values are supplied by the System Observer to any component of the system that requests them and used for further evaluation of external system behaviour.

Statistical Analyser, another internal part of the system, requests Observation Values from System Observer and performs statistical / historical analysis on the data retrieved. Based on the results of such analysis, the Statistical Analyser adjusts Observation Values in such a way that they are better tuned to represent current (or future) consumption of quality attributes, and provide such observations to any other part of the system. The Adjusted Quality Values are derived by the

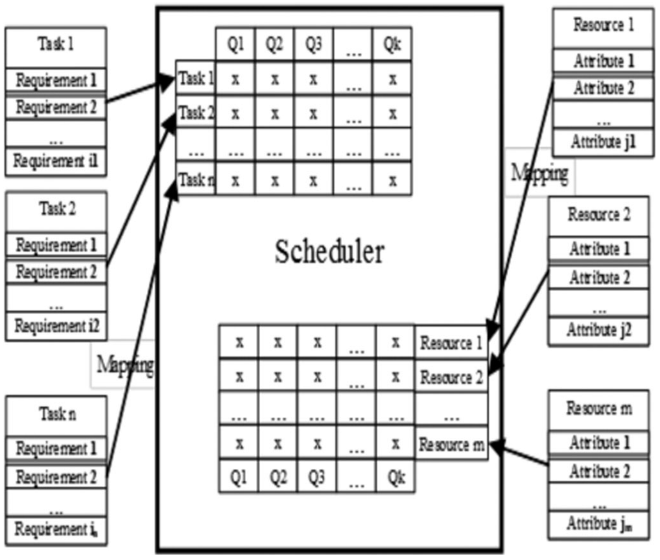


Fig. 7. A Matching mechanism.

Statistical Analyser from Normal Quality Values representing the view of the external system and are used for any further calculation while making decision.

Then, the Entity Evaluator performs evaluation of the behaviour of the external systems and makes the decisions, which are used to control their behaviours. The evaluation is carried out using ultimate Priorities, ultimate Quality Attributes and Adjusted Quality Value list as arguments. The alignment of the different visions of the system, the combination, and interaction between the different components are dictated by the BipRyt algorithm inside the Entity Evaluator component.

Finally, the Decision Maker receives the evaluation report from the Entity Evaluator. Then, it produces a list of directives (Control Instructions) that are used to regulate the behaviour of the external system.

4.3 Resource Allocation

Resource management is based on the architectural model (Fig. 6). Requests for resources, based on user requirements and the target applications, are processed and resources are allocated. A target application is considered as a set of independent tasks, *{Task1, Task2... Taskn}* each with a set of Requirements *{R1, R2 ... Rn}*, and prioritized Quality attributes *{Q1, Q2 ... Qm}*, represented by a sequence of requests for Resources *{RS1, RS2 ... RSn}* (Fig. 7). The Decider allocates tasks to resources, but if it is unable to allocate the target resource the tasks are put in a queue based on the prioritization list.

The resource allocation steps are as follow:

1. Identify and process the high-level user requirements in terms of quality attributes, together with their priority distribution.
2. Identify and gather system energy levels (CPU time, memory, bandwidth etc.).
3. Construct the matrices described earlier, including the Energy Matrix and the AHP weight of quality attributes.
4. Create distribution entries using the statistical analysis of current and historical data.

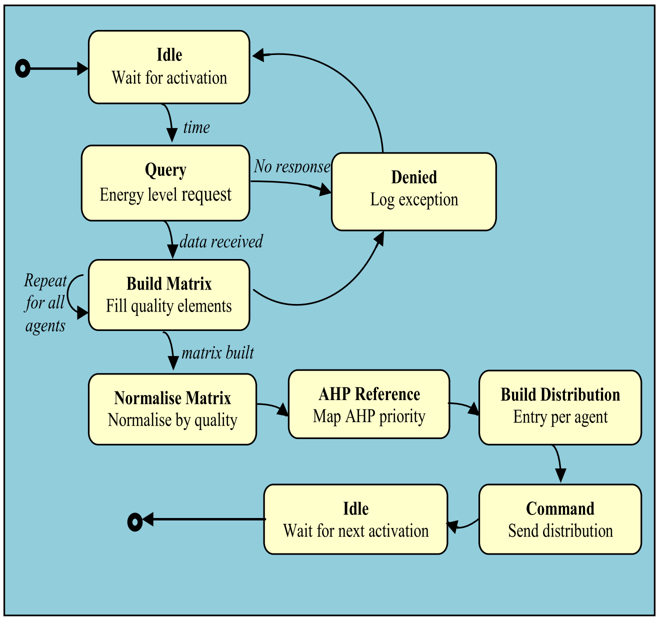


Fig. 8. Overview of decision-making processes.

5. Map the quality attributes from the requirements to the resources by the Decider, using the priority entries, and allocate tasks to available resources.
6. Check continuously for available resources or any resources that are about to saturate. If resources are about to saturate reallocated tasks accordingly.

Resource management is based on dynamic load balancing using various aspects of real-time (current) and historical statistical analysis, which produces a ranking of the available resources and their capacities (energy levels). In addition, using AHP quality attributes are prioritized and ranked. Resources that have a better balance, for example with minimum capacities that meet user quality requirements receive the corresponding proportion of the tasks. Apart from the ranking based scheduling mechanism a default scheduling policy FCFS (first come first serve) is deployed for example for similar ranking. Thus, decision making and scheduling take into account the multi-objective requirements, therefore creating a matching mechanism for a balanced tasks allocation that is both user and system centric. As shown in Fig. 7 the scheduling table contains information relating to both resources and requirements, categorized by their QoS analysis and resource management, which is based on the weighted calculation of the QoS that determines the type of resources to be matched with specific tasks.

4.4 Decision Making Processes

A state chart diagram describing a life cycle sample of the decision-making processes is depicted in Fig. 8. At the initial stage, the Decider is idle and waits for activation. When activated, the Decider broadcasts a request, as a query, to each Perceiver, which in return sends back a list of energy levels of a software agent. This process requires a Perceiver to take a number of energy levels, from the indicator reading of the gauge and push them in a list for each quality attribute per software agent. The energy levels, measureable parts, can be used to build and populate the configurable energy matrix which is then normalised, followed by the application of AHP thus reflecting user priorities. Distribution entries, using statistical analysis of current and historical data, are then created for the resource allocations and sent to individual agents. The information sent by the Perceivers is essential for helping the Decider in making decisions regarding resource allocations. The process is repeated at regular interval and is driven by user requirements.

5 Experiments

At the core of decision making in distributed system management are the scheduling algorithms used by load balancers to determine load allocation and distribution. These range from simple algorithms such as Round Robin and random choice to more sophisticated load balancers that take into account additional factors such as response times and the number of active connections. The primary aim of the experiments is to evaluate BipRyt and compare it with other decision-making mechanisms.

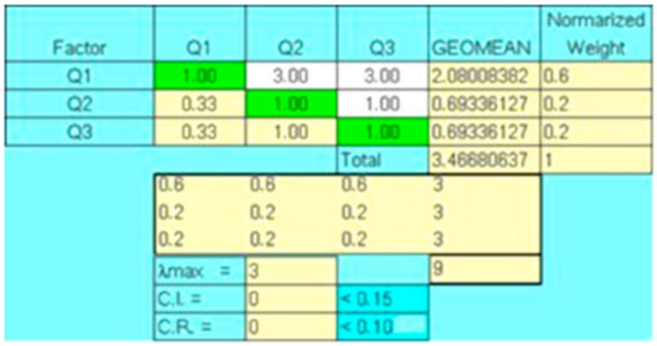
Round Robin is an even request distribution algorithm and the basic principle behind it is to distribute the request forwards (not the load) evenly. On the other hand the Response Time algorithm uses the system parameters to determine the load distribution. The Least Connection algorithm maintains a list of active connections (or requests) at any given time to each receiver. Any sub-sequential requests are being sent to a receiver that has the least amount of active requests/connections. However, the above decision making mechanisms are mainly based on a single criteria, therefore it is important to also compare BipRyt with multi-criteria mechanisms such as the Utility Model proposed in [16], [17].

The experiments are structured by order of 1) testing the adaptability of the BipRyt algorithm to the AHP trend 2) comparing the AHP trend adaptability of other load balancing algorithms, and 3) testing how BipRyt adapts to system capacity against the four algorithms. In all the experiments data packet in the form of externally generated message load is selected for the evaluation because it is 1) very dynamic to implement, 2) simple and fast to build historical data, 3) simple to build data analysis model for the evaluation, and 4) and it can be manipulated to fix its intended usage purposes, i.e., it is a controlled environment.

5.1 Experiment 1: Adaptability to AHP Trend

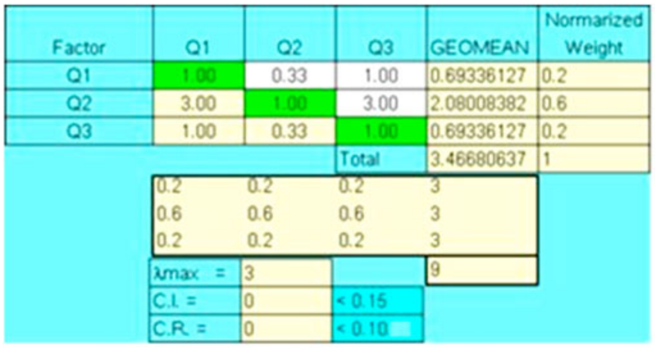
Since BipRyt uses an AHP model, that represents the user's quality expectations, to control the behaviour of an observed system, the aim of the first test is to verify if BipRyt follows the trend of the AHP configuration (settings), when making decision for a system. The configuration is defined by the user requirements from which the priorities of the quality attributes are derived.

The assumptions made for this experiment were that 1) the AHP has three quality attributes which are the number of messages per second representing throughput (Q1), the response time (Q2) and the CPU processing load (Q3); 2) the user requirements defines the priority level; 3) the overall system has 11 nodes 4) each node has three quality containers for each quality attribute mentioned above; 5) each container have three thresholds which correspond to the HRA, MRA, LRA (see Section 3.1). There is an



Factor	Q1	Q2	Q3	GEOMEAN	Normarized Weight
Q1	1.00	3.00	3.00	2.08008382	0.6
Q2	0.33	1.00	1.00	0.69336127	0.2
Q3	0.33	1.00	1.00	0.69336127	0.2
			Total	3.46680637	1
	0.6	0.6	0.6	3	
	0.2	0.2	0.2	3	
	0.2	0.2	0.2	3	
	λmax =	3		9	
	C.I. =	0	< 0.15		
	C.R. =	0	< 0.10		

Fig. 9. AHP Configuration 1.



Factor	Q1	Q2	Q3	GEOMEAN	Normarized Weight
Q1	1.00	0.33	1.00	0.69336127	0.2
Q2	3.00	1.00	3.00	2.08008382	0.6
Q3	1.00	0.33	1.00	0.69336127	0.2
			Total	3.46680637	1
	0.2	0.2	0.2	3	
	0.6	0.6	0.6	3	
	0.2	0.2	0.2	3	
	λmax =	3		9	
	C.I. =	0	< 0.15		
	C.R. =	0	< 0.10		

Fig. 10. AHP Configuration 2.

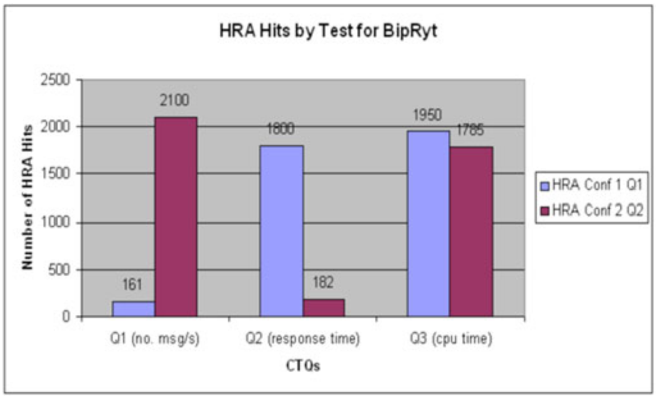


Fig. 11. BipRyt adaptability to AHP trend.

indicator mark that fluctuates across the areas illustrating how much energy a particular node has to spend for the specific quality. This means that the more hits to the high-risk area for a given quality attribute, the less capable the system is to preserve the concerned attribute.

In this experiment AHP is used to calculate the quality priority values (Normalized Weight in Fig. 9), with the obtained values are $Q1 = 0.6$, $Q2 = 0.2$, $Q3 = 0.2$ (see Sections 2.2 and 3.3). The system is then exercised for this particular configuration, and we refer to this as configuration 1. After a run of 175,650 messages received by the system, the quality priority of the AHP was changed to a new configuration, configuration 2 (Fig. 10).

The new obtained values are $Q1 = 0.2$, $Q2 = 0.6$, $Q3 = 0.2$, and the system was exercised again for another run of 175,650 incoming messages. As mentioned earlier (Section 2.2), the applied AHP process, which is based on user requirements and priorities, provides a Consistency Index (CI) and a

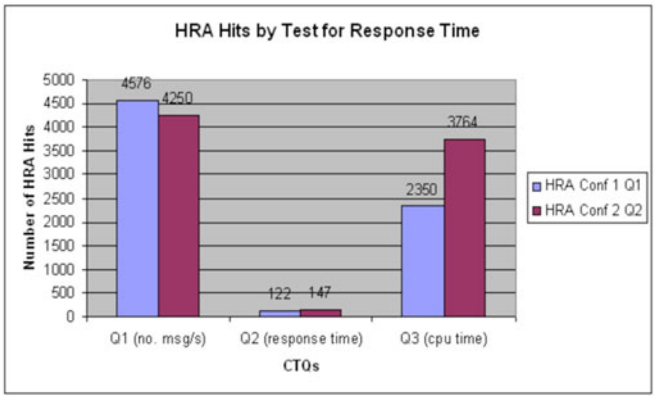


Fig. 12. Response time adaptability to AHP trend.

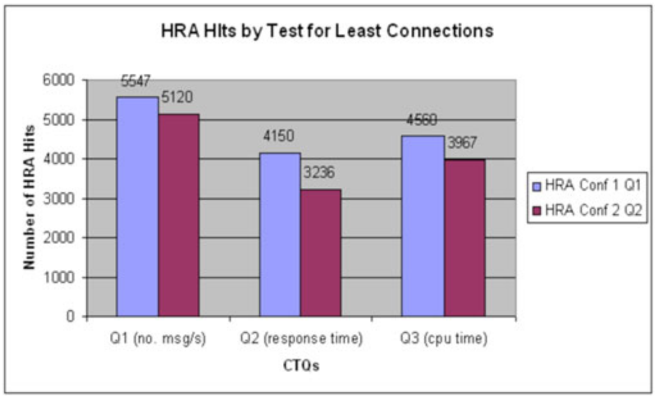


Fig. 13. Least connections adaptability to AHP trend.

Consistency Ratio (CR) to validate the consistency of the AHP results. The rule dictates that CI has to be below 15 percent and CR below 10 percent for the AHP to be truthful, which is the case for both configuration 1 and configuration 2 with $\lambda\text{max} = 3$.

As the reader can see, the test was dichotomised into two configuration setups, and for each of them, we examine the distribution of the mark of each quality attribute of each node across the 3 areas of risks. We observed that for configuration 1, where Q1 is of highest priority, the number of HRA hits for Q1, using BipRyt algorithm, is kept to 161 hits over 175,650 messages resulting to a percentage yield of 0.091 percent, which is very small. Furthermore, when the BipRyt algorithm was exercised for configuration 2, where Q2 is of highest priority, we observe that the number of HRA hits for Q2 is kept to 182 hits over 175,650 messaged resulting to a percentage yield of 0.10 percent, (see Fig. 11).

The same setup was used to test the adaptability of Response Time algorithm to the AHP trend on both configurations 1 and 2, (Fig. 12). The results show that the Response Time algorithm preserves the quality attribute Q2, response time, regardless of the AHP quality priority. For instance in the setup of configuration 1, Q1 has highest priority, yet the algorithm still preserves Q2.

The same setup was used to test the adaptability of Least Connections algorithm to the AHP trend on both configurations 1 and 2, (see Fig. 13). Since the Least Connections algorithm does not directly depend on the three quality attributes chosen for the experiments, the algorithm does not adapt to the AHP trend.

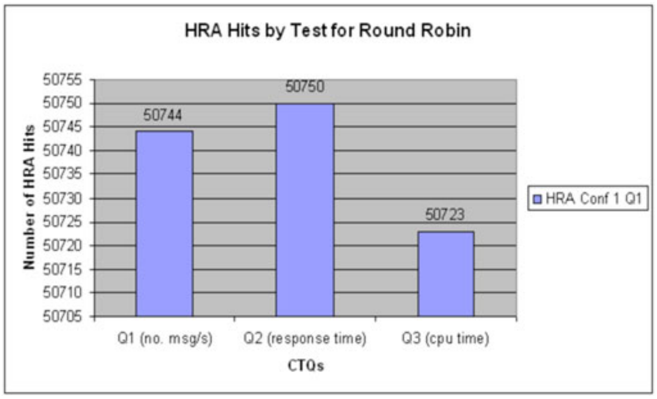


Fig. 14. Round Robin adaptability to AHP trend.

In order to make some baseline comparison a test was carried out using Round Robin. Only configuration 1 was used since the algorithm, due to its nature, does not depend on the chosen quality attributes.

As shown in Fig. 14 the number of hits to the HRA has considerably increased with Round Robin, for example the number of HRA hits for Q1 is kept to 50,744 hits over 175,650 messages resulting to a percentage yield of around 29 percent, which is almost 1/3 of the message population.

Finally, the same setup was used to test the adaptability of the multi-criteria Utility Model [16], [17] to the AHP trend on both configurations 1 and 2, (see Fig. 15). Since the Utility Model algorithm focuses on maximizing global system utilization, while meeting the minimum needs of user requirements, the algorithm behaviour is similar, to a certain extent, to Response Time; however, it adapts better to Q1 but worse to Q2. Overall, it does adapt only partially to the AHP trend.

Overall the results show that BipRyt adapts better to AHP trends than the other algorithms. However, since the Response Time algorithm was designed specifically to operate on response time (Q2) it preserves Q2 better than the BipRyt algorithm for configuration 2 where Q2 is of highest priority. In fact, BipRyt with configuration $Q1 = 0$, $Q2 = 1$, $Q3 = 0$, behaves in a similar way to Response time. Moreover, Response Time's overall number of hits is still much lower than that of Round Robin, since response time (Q2) does positively influence the other quality attributes, for example it does indirectly reflect CPU time due to the fact that if a given node responds faster to a request, this means that the CPU load is lower.

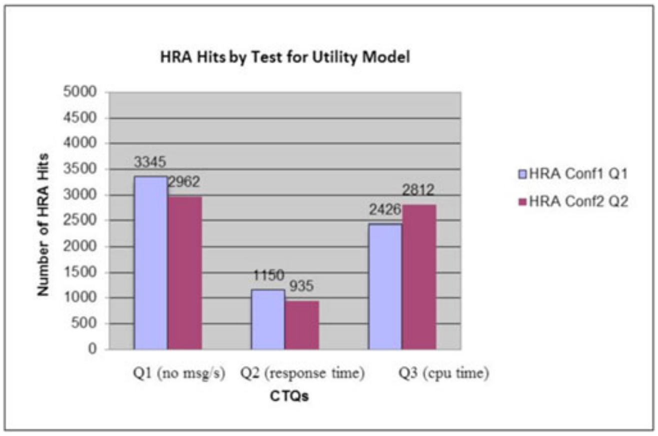


Fig. 15. Utility model adaptability to AHP trend.

TABLE 3
System Configuration

Nodes	Queue System	Buffer Size
0	S0Q0	940,000
1	S1Q1	860,000
2	S2Q2	780,000
3	S3Q3	600,000
4	S4Q4	520,000
5	S5Q5	470,000
6	S6Q6	360,000
7	S7Q7	220,000
8	S8Q8	110,000
9	S9Q9	90,000
10	S10Q10	80,000

Using Least Connections the distribution of the HRA hits resembles the Round Robin, but in a lower order of magnitude. The reason for lower hits is again, due to the fact that Least Connections may also influence attributes such as CPU time and response time, but not always. For example, when the load is distributed based on least connections, the next requests are forwarded to the node with the least connections, hence less messages are being processed on that node, preserving CPU time. As a result, the number of HRA hits for CPU time is reduced.

While the Utility Model follows to a certain degree the Response time, it is better than Response time at meeting overall criteria. But, BipRyt still adapt better than the Utility Model to user requirements and AHP trend in both configuration 1 and 2. In summary, the BipRyt algorithm maximises the opportunity for a system to conform to the user's priorities for example in the first configuration Q1, which has the highest priority, has the best percentage yield. However, this changes in configuration 2 when Q2 has the highest priority and the best yield. It is clear that similar results will be achieved if we deploy another configuration $Q1 = 0.2$, $Q2 = 0.2$, $Q3 = 0.6$, with now Q3 yielding the best result this time.

5.2 Experiment 2: Adaptability to the Capacity of Infrastructure

The aim of this experiment is to check the overall adaptability of BipRyt, Response Time, Least Connections, Round Robin load and the Utility Model balancing algorithms to the Capacity of the Infrastructure, and to analyze how each of them distributes the load with heterogeneous nodes capacities. To achieve this we designed 11 queues, with different configurations and different capacities (Table 3). Each queue has different buffer size, which represents the memory capacity of the node. We incorporated a processing function to emulate latency within the logic of each queue, which is triggered when a message is loaded into the buffer. We also simulated input/output by defining write requests to a mySQL database when a message is being processed, and we deployed the queue applications on 11 nodes.

As before we started by obtaining the quality priority from the AHP. So in the experiment we have three quality attributes, which are CPU Time (Q1), memory usage (Q2) and number of database I/O (Q3). The obtained quality

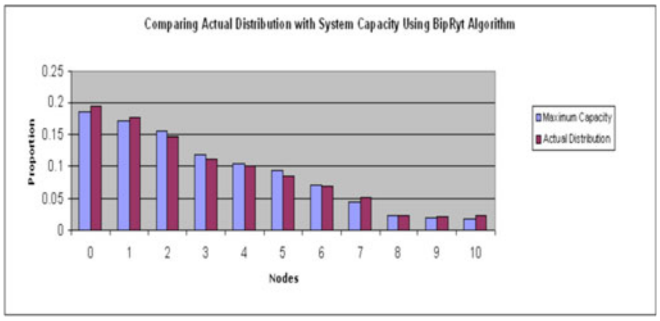


Fig. 16. BipRyt distribution versus system capacity.

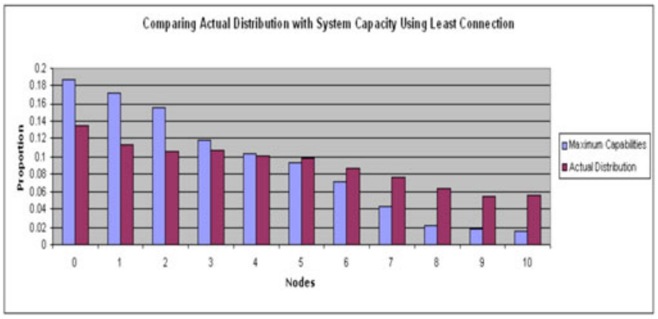


Fig. 18. Least connections distribution versus system capacity.

priorities are Q1 = 0.3, Q2 = 0.6 and Q3 = 0.1, and provisioned the BipRyt algorithm.

First, the system runs whilst the maximum buffer capacity of individual queues per nodes is adjusted until there is no hit to the high-risk area of the quality container, given a fix number of messages per second. Next, we performed the calibration of the system overall capacity by adding the individual load capacity together. Having calibrated the nodes capacities, the system runs for 10 minutes starting with the BipRyt algorithm, then, Response Time, Least Connections, Round Robin and finishing with the Utility Model. At the end of each run, the number of hits to the high-risk areas for the quality attributes and buffer population, which defines the memory available, or message load for each node are recorded. In all the experiments the maximum capacity of memory and the actual distribution of messages are represented by the normalized values of the proportion of buffer sizes and number of messages in relation to their total aggregated values.

When the system was exercised with the BipRyt algorithm, in 10 minutes, 464,038 messages were distributed to the nodes. As it can be observed in Fig. 16, BipRyt intelligently balances the incoming load as per maximum capability of the nodes. The actual distribution follows the trend of the node capacity, implying that the system is efficient with negligible number of node message starvation and little packet loss. Packet loss is reached when the number of message sent to a node is greater than the buffer size of a given queue.

When the system was exercised with the Response Time algorithm, in 10 minutes, 1,494,335 messages were distributed to the nodes. The graph in Fig. 17 shows that the Response Time algorithm does attempt to follow the trend of the nodes' maximum capacities. However, the difference between the nodes' maximum capabilities and the actual distribution of Response Time is still a bit larger than the difference of the BipRyt algorithm, meaning that there is still more occurrences of message starvation and packet loss. This is because the BipRyt algorithm directly preserves the message load in terms of the quality attribute memory available. Whereas the Response Time algorithm indirectly preserves the message load by managing the quality attribute response time which indirectly influences the quality attribute memory usage.

When the system was exercised with the Least Connections algorithm, in 10 minutes, 1,583,827 messages were distributed to the nodes. The graph in Fig. 18 shows that the trend of the actual distribution for the Least Connections does not follow the trend of the nodes' maximum capacities. This is because Least Connections is not directly influenced by the three quality attributes defined for this experiment. Unlike Response Time, Least Connections has little impact on memory usage and CPU Time. Indeed Least Connections is not a good measure for CPU Time and load, since one node may have five connections of 2 megabytes each and a second one, having two connections of 20 megabytes load each and due to the fact that the algorithm base its decision on the number of connections rather than load per connections, it does not provide a good measure on the quality attributes CPU load or memory usage. Hence the Least Connections failed to adapt its distribution of message to the maximum capabilities of the nodes, but still provides a better solution than Round Robin.

As Fig. 19 illustrates, due to its simplistic nature, the Round Robin algorithm distributes the load uniformly regardless to the quality attributes and the maximum capacities of the nodes. So at some point in time, Round Robin will starve some the nodes with highest capacity and overload the weakest. In such an environment, wherein nodes have different load capacities, Round Robin is very inefficient especially for packet loss. In 10 minutes, with Round Robin,

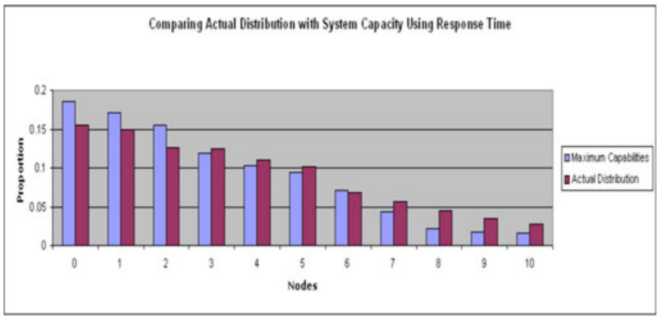


Fig. 17. Response time distribution versus system capacity.

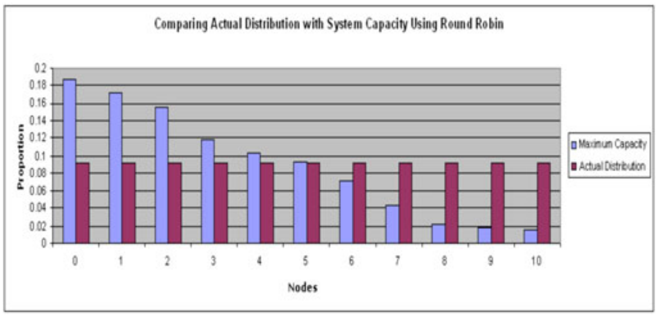


Fig. 19. Round robin distribution versus system capacity.

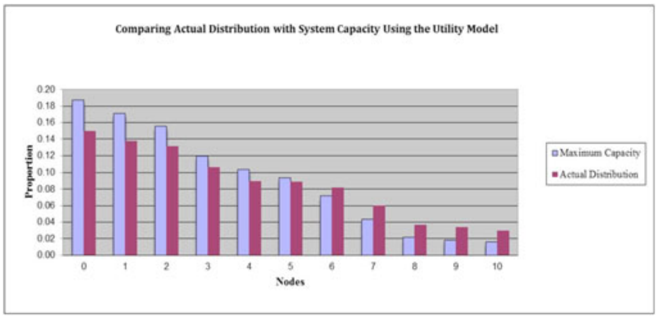


Fig. 20. The utility model distribution versus system capacity.

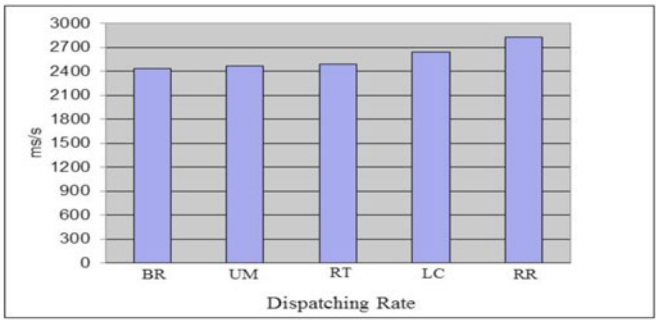


Fig. 21. Dispatching rate of the five algorithms.

1,698,771 messages were distributed to the nodes but many of the packets where rejected by the low performing nodes.

Finally, the same setup was used to check the overall adaptability of the multi-criteria Utility Model to the Capacity of the Infrastructure. As shown in Fig. 20 the Utility Model follows the system capacity trends, however, it is still less adaptable than BipRyt, which has lower message rejection and node starvation.

A number of further experiments were carried out to evaluate and compare the performance of the different algorithms starting with Fig. 21 showing the number of messages per second, representing throughput, that were dispatched by the different load balancers. This shows clearly that Round Robin (RR) has the highest rate, since it has the least overheads due to its simplicity, followed by Least Connections (LC), Response time (RT), Utility Model (UM) and BipRyt (BR).

However, message rejection rates also follow the same order i.e., Round Robin has the highest rejection rate and BipRyt the lowest (Fig. 22). Clearly, similar results can be obtained when node starvation rate is considered.

Moreover, in order to evaluate the true cost associated with each of the algorithms rejected messages have to be taken into account and processed. Thus, in the next experiment the algorithms are evaluated with rejected messages returning back to the load balancer buffer queue for processing (Fig. 23). This shows that, despite the additional processing time noted in Fig. 21, when rejection rate is taken into account BipRyt shows the best performance followed by the Utility Model, Response Time, Least Connections and Round Robin. This reflects the benefits of the algorithm particularly in heterogeneous environment.

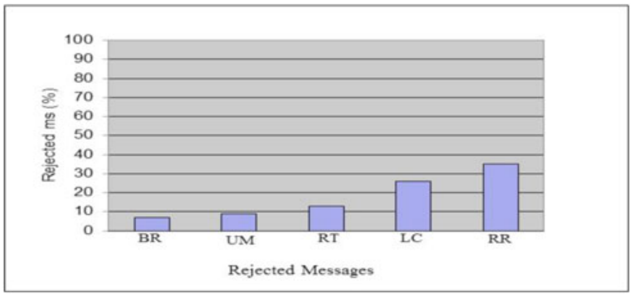


Fig. 22. Rejected messages rate of the five algorithms.

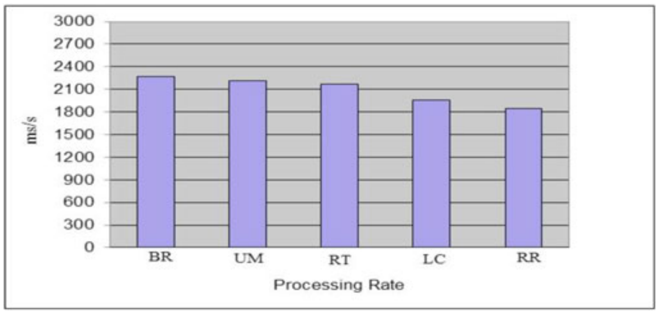


Fig. 23. Processing rate of the five algorithms.

6 Conclusion

This paper presented a multi-criteria decision making mechanism, which has been designed for the management, distribution and optimization of systems resources. The novelty of the BipRyt algorithm lies in the ability to use a quality model based on a multitude of quality requirements. The algorithm was incorporated into a generic architecture, which can be integrated into any load balancer or scheduler components for distributed systems.

Moreover, during the design and implementation of the BipRyt algorithm, we observed that techniques such as the Analytical Hierarchy Process provide instructions that can be automated through a sequence of actions. Hence, the mechanism has been constructed to represent both run time quality assurance and dynamic quality enforcer. BipRyt takes into account the user's perspective of the quality model, and ensures the system follows the quality trend of the user. Thus, it preserves the quality attributes while avoiding nodes overloading or starvation.

To validate the functionality of BipRyt in enforcing a defined quality model, we presented a case study that implements this algorithm as a decision-making mechanism. We tested BipRyt against the Round Robin, Response Time, Least Connections and Utility Model strategies. The results showed that BipRyt efficiently distributed the workload especially in heterogeneous network, with different capability nodes. However, the use of multiple deciders and the impact of larger and potentially conflicting quality requirements in non-controlled environments will be considered in future work.

References

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Comput. Syst.*, vol. 25, no. 6, pp. 599–616, 2009.
[2] T. Rings, et al., "Grid and cloud computing: Opportunities for integration with the next generation network," *J. Grid Comput.*, vol. 7, no. 3, pp. 1572–9814, 2009.
[3] M. Armbrust, et al., "Above the clouds: A Berkeley view of cloud computing," EECS Department, University of California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2009-28, 2009.

[4] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*. Englewood Cliffs, NJ, USA: Prentice Hall, 2005.

[5] N. M. Josuttis, *SOA in Practice*. Philadelphia, PA, USA: O'Reilly, 2007

[6] G. Horgan and S. Khaddaj, "Use of an adaptable quality model in a production support environment," *Systems Softw.*, vol. 82, pp. 730–738, 2009.

[7] S. Khaddaj, "Quality of service issues in distributed component based environments," *Algorithms Comput. Technol.*, vol. 4, pp. 523–531, 2010.

[8] S. Khaddaj and G. Horgan, "The evaluation of software quality factors in very large information systems," *E-J. Inf. Syst. Eval.*, vol. 7, pp. 43–48, 2004.

[9] S. Khaddaj and H. Nguyen, "Cloud computing: The management of service level agreements," in *Proc. Int. Conf. Semantic E Business and Enterprise Computing. SEEC 2010*, Excel India Publishers, 2010, pp. 24–31.

[10] V. Stantchev and C. Schröpfer, "Negotiating and Enforcing QoS and SLAs in Grid and Cloud Computing," in *Proc. 4th Int. Conf. Adv. Grid Pervasive Comput.*, 2009, pp. 25–35.

[11] Z. Wu, X. Liu, Z. Ni, D. Yuan, and Y. Yang, "A market-oriented hierarchical scheduling strategy in cloud workflow systems," *J. Supercomputing*, vol. 63, no. 1, pp. 1–38, 2011.

[12] B. Q. Cao, B. Li, and Q. M. Xia, "A service-oriented QoS-assured and multi-agent cloud computing architecture," *Cloud Computing*. Berlin, Germany: Springer, 2009.

[13] J. Al-Ali, A. Hafid, F. Rana, and W. Walker, "An approach for quality of service adaptation in service-oriented grids," *Concurrency Comput. Practice Experience*, vol. 6, no. 5, pp. 401–412, 2004.

[14] D. A. Menascé, H. Ruan, and H. Gomaa, "QoS management in service-oriented architectures," *Performance Eval.*, vol. 64, no. 7–8, pp. 646–663, 2007.

[15] H. Nguyen and S. Khaddaj, "A QoS based load balancing framework for large scale elastic distributed systems," in *Proc. 10th Int. Symp. Distrib. Comput. Appl. Bus. Eng. Sci.*, 2011, pp. 135–140.

[16] C. Li and L. Li, "Utility-based QoS optimisation strategy for multi-criteria scheduling on the grid," *J. Parallel Distrib. Comput.*, vol. 67, pp. 142–153, 2007.

[17] G. Tang, H. Li, and S. Yao, "The multi-dimensional QoS resources optimization based on the grid banking model," in *Proc. 2nd Int. Conf. High Perform. Comput. Appl.*, 2009, pp. 369–376.

[18] R. Buyya, *High Performance Cluster Computing: Architectures and Systems, Volume 1*. Delhi, India: Prentice Hill PTR, 1999.

[19] B. A. Shirazi, K. M. Kavi, and A. R. Hurson, *Scheduling and Load Balancing in Parallel and Distributed Systems*. Los Alamitos, CA, USA: Wiley-IEEE Computer Society Press, 1995.

[20] J. Westbrook, "Load balancing for response time," *Annu. Eur. Symp. Algorithms*, 1995, pp. 355–368.

[21] F. Dong and S. G. Akl, "Scheduling algorithms for grid computing: state of the art and open problems," School Comput., Queen's Univ., Kingston, ON, Canada, Tech. Rep. 2006-504, 2006.

[22] T. Casavant and J. Kuhl, "A taxonomy of scheduling in general-purpose distributed computing systems," *IEEE Trans. Softw. Eng.*, vol. 14, no. 2, pp. 141–154, Feb. 1988.

[23] C. Xu and F. C.M. Lau, *Load Balancing in Parallel Computers: Theory and Practice*. New York, NY, USA: Springer, 1997.

[24] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual infrastructure management in private and hybrid clouds," *IEEE Internet Comput.*, vol. 13, no. 2, pp. 14–22, Nov. 2009.

[25] M. Randles, D. Lamb, and A. Taleb-Bendiab, "A comparative study into distributed load balancing algorithms for cloud computing," in *Proc. IEEE 24th Int. Conf. Adv. Inf. Netw. Appl.*, 2010, pp. 551–556.

[26] A. Singh, M. Korupolu, and D. Mohapatra, "Server-storage virtualization: Integration and load balancing in data centers," in *Proc. ACM/IEEE Conf. Supercomputing*, 2008, pp. 53–65.

[27] Y. Fang, F. Wang, and J. Ge, "A task scheduling algorithm based on load balancing in cloud computing," in *Proc. Int. Conf. Web Inf. Syst. Mining*, 2010, vol. 6318, pp. 271–277.

[28] N. Haider, S. Khaddaj, M. Wilby, and D. Vvedensky, "Parallel Monte Carlo simulations of epitaxial growth," *Comput. Phys.*, vol. 9, no. 1, pp. 85–96. 1995.

[29] B. Mrohs, "OWL-SF—A distributed semantic service framework," in *Proc. Workshop Context Awareness Proactive Syst.*, 2005, pp. 67–79.

[30] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Proc. Grid Comput. Environ. Workshop*, 2008, pp. 1–10.

[31] M. Gabbrielli, *Programming Languages: Principles and Paradigms*. Berlin, Germany: Springer, 2010.

[32] M. V. Zelkowitz, *Advances in Computers: New Programming Paradigms*. Cambridge, MA, USA: Academic Press, 2005.

[33] H. J. Braun, et al., "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *J. Parallel Distrib. Comput.*, vol. 61, no. 6, pp. 810–837, 2001.

[34] M. H. Willebeek-LeMair and A. P. Reeves, "Strategies for dynamic load balancing on highly parallel computers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 9, pp. 979–993, Sep. 1993.

[35] M, Arora, S. K. Das, and R. Biswas, "A decentralized scheduling and load balancing algorithm for heterogeneous grid environments," in *Proc. ICPPW'02*, 2002, pp. 499–505.

[36] G. Sabin, R. Kettimuthu, A. Rajan, and P. Sadayappan, "Scheduling of parallel jobs in a heterogeneous multi-site environment," in *Proc. 9th Int. Workshop Job Scheduling Strategies Parallel Process.*, 2003, pp. 87–104.

[37] S. Y. You, H. Y. Kim, D. H. Hwang, and S. C. Kim, "Task scheduling algorithm in GRID considering heterogeneous environment," in *Proc. Int. Conf. Parallel Distrib. Process. Tech. Appl.*, 2004, pp. 240–245.

[38] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "Adaptive load sharing in homogeneous distributed systems," *IEEE Trans. Softw Eng.*, vol. 12, no. 5, pp. 662–675, May 1986.

[39] A. M. Alakeel, "A Guide to dynamic load balancing in distributed computer systems," *Int. J. Comput. Sci. Netw. Secur.*, vol. 10, no. 6, pp. 153–160, Jun. 2010.

[40] M. Zaki, W. Li, and S. Parthasarathy, "Customized dynamic load balancing for a network of workstations," *J. Parallel Distrib. Comput.*, vol. 43, pp. 156–162, 1997.

[41] T. Kunz, "The influence of different workload descriptions on a heuristic load balancing scheme," *IEEE Trans. Softw. Eng.*, vol. 17, no. 7, pp. 725–730, Aug. 1991.

[42] J. Cao, X. Wang, and S. K. Das, "A framework of using cooperating mobile agents to achieve load sharing in distributed web server groups," *Future Generation Comput. Syst.*, vol. 20, pp. 591–603, 2004.

[43] D. Grosu and A. T. Chronopoulos, "Noncooperative load balancing in distributed systems," *J. Parallel Distrib. Comput.*, vol. 65, no. 9, pp. 1022–1034, 2005.

[44] A. Y. Zomaya and Y. The, "Observations on using genetic algorithms for dynamic load-balancing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 9, pp. 899–911, Sep. 2001.

[45] D. Ong and S. Khaddaj, "Intelligent framework for the management of distributed architectures," in *Proc. 11th ACIS Int. Softw. Eng. Artif. Intell. Netw. Parallel/Distrib. Comput.*, 2010, pp. 187–192.

[46] Y. C. Lee and A. Y. Zomaya, "Energy conscious scheduling for distributed computing systems under different operating conditions," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 8, pp. 1374–1381, Aug. 2011.

[47] V. Stantev, "Effects of replication on web service performance in WebSphere," Int. Comput. Sci. Inst., Univ. California, Berkeley, CA, USA, Tech. Rep. TR-08-003, 2008.

[48] R. Cakubescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic QoS management and optimization in service-based systems," *IEEE Trans. Softw. Eng.*, vol. 37, no. 3, pp. 387–409, May/Jun. 2011.

[49] D. Adagna and B. Pernici, "Adaptive service composition in flexible processes," *IEEE Trans. Softw. Eng.*, vol. 33, no. 6, pp 369–384, Jun. 2007.

[50] W. Ma, L. Liu, H. Xie, H. Zhang, and J. Yin, "Preference model driven services selection," *Adv. Inf. Syst. Eng.*, vol. 5565, pp. 216–230, 2009.

[51] H. Xie, L. Liu, and J. Yang, "i*-prefer: Optimizing requirements elicitation process based on actor preferences," in *Proc. 2009 ACM Symp. Appl. Comput.*, 2009, pp. 347–354.

[52] A. Streit, "Evaluation of an unfair decider mechanism for the self-tuning dynP job scheduler," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2004, pp. 1–11.

[53] T. L. Saaty, *The Analytic Hierarchy Process*. New York, NY, USA: McGraw-Hill, 1980.

[54] J. Karlsson, "Software requirements prioritizing," in *Proc. 2nd Int. Conf. Requirements Eng.*, 1996, pp. 110–116.

[55] E. H. Forman and S. I. Gass, "The analytic hierarchy process: An exposition," *Oper. Res.*, vol. 49, pp. 469–486, 2001.

[56] R. Buyya, "Economic models for resource management and scheduling in grid computing," *Concurrency Comput.*, vol. 14, 1507–1542, 2002.

[57] M. Maurer, I. Breskovic, V. C. Emeakaroha, and I. Brandic, "Revealing the MAPE loop for the autonomic management of cloud infrastructures," in *Proc. IEEE Symp. Comput. Commun.*, 2011, pp. 147–152.

[58] N. Bansal, A. Awasthi, and S. Bansal, "Task scheduling algorithms with multiple factor in cloud computing environment," *Adv. Intell. Syst. Comput.*, vol. 433, pp 619–627, 2016.

[59] M. Masdari, S. ValiKardan, Z. Shahi, and S. I. Azar, "Towards workflow scheduling in cloud computing: A comprehensive analysis," *J. Netw. Comput. Appl.*, vol. 66, pp. 64–82, 2016.

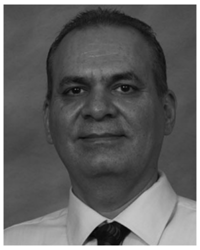
[60] S. Singh and I Chana, "A survey on resource scheduling in cloud computing: Issues and challenges," *J. Grid Comput.*, vol. 14, no. 2, pp. 217–264, 2016.

[61] A. Abdelmaboud, D. N. A. Jawawi, I. Ghani, A. Elsafi, and B. Kitchenham, "Quality of service approaches in cloud computing: A systematic mapping study," *J. Syst. Softw.*, vol. 101, pp. 159–179, 2015.

[62] C. Tse-Shih, T.-W. Huang, L.-C. Yin, Y.-L. Chen, and Y.-F. Ciou, "Platform-as-a-service architecture for parallel video analysis in clouds" in *Proc. Int. Comput. Symp. Adv. Intell. Syst. Appl.*, 2013, vol. 2, pp. 619–626.

[63] C. Bunch, V. Arora, N. Chohan, C. Krintz, S. Hegde, and A. Srivastava, "A pluggable autoscaling service for open cloud PaaS systems," in *Proc. IEEE/ACM 5th Int. Conf. Utility Cloud Comput.*, 2012, pp. 191–194.

[64] M. Boniface, et al., "Platform-as-a-service architecture for real-time quality of service management in clouds," in *Proc. 5th Int. Conf. Internet Web Appl. Serv.*, 2010, pp. 155–160.



Souheil Khaddaj received the PhD degree from the Centre of Parallel Computing, Queen Mary College. He is a professor of computer science in the School of Computer Science and Mathematics, Kingston University - London, where he leads the Component & Distributed Systems Research Group (CODIS). His research interests include distributed computing, service orientation and big data. He has been involved in the development of novel technologies for numerous scientific and business applications. His research interests also include advanced software engineering techniques and quality assurance. He has been involved in a large number of national and international research projects and various industrial partnerships. He also chaired many international conferences and he has been a keynote speaker at numerous international events. He has authored/co-authored more than 200 technical papers and he has also edited/co-edited a number of books and special issues.

Bippin Makoond received the PhD degree from Kingston University. Currently he is the managing director and founder of ZDLC Cognizant Technology Solutions, who functions as the Global Innovation Lead for the company's Banking and Financial Services practice. His research interests include quality engineering and service oriented distributed systems. He published many papers in Journals and Conferences. He also holds three patents within the domain of Wireless Distributed Systems and is a visiting scholar at Kingston University.