Cumhuriyet Üniversitesi Fen Fakültesi
Fen Bilimleri Dergisi (CFD), Cilt:36, No: 3 Özel Sayı (2015)
ISSN: 1300-1949

Cumhuriyet University Faculty of Science
Science Journal (CSJ), Vol. 36, No: 3 Special Issue (2015)
ISSN: 1300-1949

Scheduling Problems For Cloud Computing

Anis VOSOOGH[1], Reza NOURMANDİ-POUR[2,*]

[1]*Department of computer engineering, Sirjan Science and research branch, Islamic azad university, Sirjan, Iran & Department of computer engineering, Sirjan branch, Islamic azad university, Sirjan, Iran*

[2]*Department of computer engineering, Sirjan branch, Islamic azad university, Sirjan, Iran*

Received: 01.02.2015; Accepted: 05.05.2015

Abstract. Cloud computing, the long-held dream of computing as a utility, has the potential to transform a large part of the IT industry, making software even more attractive as a service and shaping the way in which hardware is designed and purchased. From the theoretical aspect, we mainly accomplish three research issues. Firstly, we solve the resource allocation problem in the user-level of cloud scheduling. We propose game theoretical algorithms for user bidding and auctioneer pricing.With Bayesian learning prediction, resource allocation can reach Nash equilibrium among non-cooperative users even though common knowledge is insufficient. Secondly, we address the task scheduling problem in the system-level of cloud scheduling. We prove a new utilization bound to settle on-line schedulability test considering the sequential feature of MapReduce. We deduce the relationship between cluster utilization bound and the ratio of Map to Reduce. This new schedulable bound with segmentation uplifts classical bound which is most used in industry.

Keywords: Cloud computing, Scheduling problems, Cloud evolution, Cloud service, Cloud characteristics.

1. INTRODUCTION

1.1 Research Background

Cloud computing is everywhere. When we open any IT magazines, websites, radios or TV channels, ”cloud” will definitely catch our eye. Today’s most popular social networking, email, document sharing and online gaming sites, are hosted on a cloud. More than half of Microsoft developers are working on cloud products. Even the U.S government intends to initialize cloud-based solutions as the default option for federal agencies of 2012. Cloud computing makes software more attractive as a service, and shapes the way in which IT hardware is purchased. Predictably, it will spark a revolution in the way organizations provide or consume information and computing.

Datacenters, behaving as ”cloud providers”, are computing infrastructures which provide many kinds of agile and effective services to customers. A wide range of IT companies including Amazon, Cisco,Yahoo, Salesforce, Facebook, Microsoft and Google have their own datacenters and provide pay-as-you-go cloud services. Two different but related types of cloud service should be distinguished first. One is on-demand computing instance, and the other is on-demand computing capacity. Equipped with similar machines, datacenters can scale out by providing additional computing instances, or can support data- or compute-intensive applications via scaling capacity.

Amazon’s EC2 and Eucalyptus are examples of the first category, which provides computing instances according to needs. The datacenters instantly creat virtualized instances and give the response. The virtualized instance might consist of processors running at different speeds and storage that spans different storage systems at different locations. Therefore, virtualization is an

* Corresponding author. *Email address*: *noormandi_r@iausirjan.ac.ir*

Special Issue: The Second National Conference on Applied Research in Science and Technology essential characteristic of cloud computing, through which applications can be executed independently without regard for any particular configuration.

Google and Yahoo belong to the second category. In these datacenters, the need of processing large amounts of raw data is primarily met with distributed and parallel computing, and the data can be moved from place to place and assigned changing attributes based on its lifecycle, requirements, and usefulness. One core technology is MapReduce, a style of parallel programming model supported by capacity-on-demand clouds. It can compute massive data in parallel on a cloud.

1.2 Challenges And Motivations

Cloud computing is still in its infancy, but it has presented new opportunities to users and developers who can benefit from economies of scales, commoditization of assets and conformance to programming standards. Its attributes such as scalability, elasticity, low barrier to entry and a utility type of delivery make this new paradigm quickly marketable.

However, cloud computing is not a catholicon. The illusion of scalability is bounded by the limitations that cloud providers place on their clients. Resource limits are exposed at peak conditions of the utility itself. For example, bursting spring festival messages lead to outage for telecom operators, so they have to set limits on the number of short messages before New Year Eve. The same problem appears in cloud computing. These outages will happen on peak computing days such as the day when Internet Christmas sales traditionally begin. Additionally, Internet is one basis of the cloud, so an unavoidable issue is that network bottlenecks often occur when large data is transferred. In that case, the burden of resource management is still in the hands of users, but the users usually have limited management tools and permission to deal with these problems [1].

2. CLOUD COMPUTING OVERVIEW

2.1. Introduction

This chapter begins with a general introduction of cloud computing, followed by the retrospect of cloud evolution history and comparison with several related technologies. Through analyzing system architecture, deployment model and service type, the characteristics of cloud computing are concluded from technical, functional and economical aspects. After that, current efforts both from commercial and research perspectives are presented in order to capture challenges and opportunities in this domain.

2.1.1. Cloud Definitions

Since 2007, the term Cloud has become one of the most buzz words in IT industry. Lots of researchers try to define cloud computing from different application aspects, but there is not a consensus definition on it. Among the many definitions, we choose three widely quoted as follows:

• **I. Foster** [2]: “A large-scale distributed computing paradigmthat is driven by economies of scale, in which a pool of abstracted virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over Internet.” As an academic representative, Foster focuses on several technical features that differentiate cloud computing from other distributed computing paradigms. For example, computing entities are virtualized and delivered as services, and these services are dynamically driven by economies of scale.

• **Gartner** [3]: “A style of computing where scalable and elastic IT capabilities are provided as a service to multiple external customers using Internet technologies.” Garter is an IT consulting company, so it examines qualities of cloud clouding mostly from the point of view of industry. Functional characteristics are emphasized in this definition, such as whether cloud computing is scalable, elastic, service offering and Internet based.

• **NIST**[4]: “Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.” Compared with other two definitions, U.S. National Institute of Standards and Technology provides a relatively more objective and specific definition, which not only defines cloud concept overall, but also specifies essential characteristics of cloud computing and delivery and deployment models.

2.1.2. Deployment Models

Clouds are deployed in different fashions, depending on the usage scopes. There are four primary cloud deployment models.

• **Public cloud** is the standard cloud computing paradigm, in which a service provider makes resources, such as applications and storage, available to the general public over Internet. Service providers charge on a fine-grained utility computing basis. Examples of public clouds include Amazon Elastic Compute Cloud (EC2), IBM’s Blue Cloud, Sun Cloud, Google AppEngine and Windows Azure Services Platform.

• **Private cloud** looks more like a marketing concept than the traditional mainstreamsense. It describes a proprietary computing architecture that provides services to a limited number of people on internal networks. Organizations needing accurate control over their data will prefer private cloud, so they can get all the scalability, metering, and agility benefits of a public cloud without ceding control, security, and recurring costs to a service provider. Both eBay and HP CloudStart yield private cloud deployments.

• **Hybrid cloud** uses a combination of public cloud, private cloud and even local infrastructures, which is typical for most IT vendors. Hybrid strategy is proper placement of workloads depending upon cost and operational and compliance factors. Major vendors including HP, IBM, Oracle and VMware create appropriate plans to leverage a mixed environment, with the aim of delivering services to the business. Users can deploy an application hosted on a hybrid infrastructure, in which some nodes are running on real physical hardware and some are running on cloud server instances.

2.2. Cloud Evolution

Although the idea of cloud computing is not new, it has rapidly become a new trend in the information and communication technology domain and gained significant commercial success over past years. No one can deny that cloud computing will a play pivotal role in the next decade. Why cloud computing emerges now, not before? This section looks back on the development history of cloud computing.

2.2.1. A Brief History

Along with the maturity of objective conditions (software, hardware), plenty of existing technologies, results, and ideas can be realized, updated, merged and further developed. Amazon played a key role in the development of cloud computing by initially renting their datacenter to external customers for the use of personal computing. In 2006, they launched Amazon EC2 and S3 on a utility computing basis. After that, several major vendors released cloud solutions one after another, including Google, IBM, Sun, HP, Microsoft, Forces.com, Yahoo and so on. Since 2007, the number of trademarks covering cloud computing brands, goods and services has increased at an almost exponential rate. Cloud computing is also a much favored research topic. In 2007, Google, IBM and a number of universities announced a research project, Academic Cloud Computing Initiative (ACCI), aiming at addressing the challenges of large-scale distributed computing. Since 2008, several open source projects have gradually appeared. For example, Eucalyptus is the first API-compatible platform for deploying private clouds. OpenNebula deploys private and hybrid clouds and federates different modes of clouds.

2.3. Cloud Service

As an underlying delivery mechanism, cloud computing ability is provisioned as services, basically in three levels: software, platform and infrastructure [5].

2.3.1. Software As A Service

Software as a Service (SaaS) is a software delivery model in which applications are accessed by a simple interface such as a web browser over Internet. The users are not concerned with the underlying cloud infrastructure including network, servers, operating systems, storage, platform, etc. This model also eliminates the needs to install and run the application on the local computers. The term of SaaS is popularized by Salesforce.com, which distributes business software on a subscription basis, rather than on a traditional on-premise basis. One of the best known is the solution for its Customer Relationship Management (CRM). Now SaaS has now become a common delivery model for most business applications, including accounting, collaboration and management. Applications such as social media, office software, and online games enrich the family of SaaS-based services, for instance, web Mail, Google Docs, Microsoft online, NetSuit, MMOG Games, Facebook, etc.

2.3.2. Platform As A Service

Platform as a Service (PaaS) offers a high-level integrated environment to build, test, deploy and host customer-created or acquired applications. Generally, developers accept some restrictions on the type of software that can write in exchange for built-in application scalability. Customers of PaaS do not manage the underlying infrastructure as SaaS users do, but control over the deployed applications and their hosting environment configurations.

3. SCHEDULING PROBLEMS FOR CLOUD COMPUTING

3.1. Introduction

This chapter outlines the scheduling problems arising from cloud computing. Concerned theories including former expressions of problems, algorithms, complexity and schematic methods are briefly introduced. Then scheduling hierarchy in cloud datacenter is presented, by splitting scheduling problem into user-level and system-level. The former focuses on resource provision issues between providers and customers, which are solved by economic models. The latter refers to meta-task execution, a sub-optimal solution of which is given by heuristics to speed up the process of finding a good enough answer. Moreover, real-time scheduling attracts our attention. Different from economic and heuristic strategies, priority scheduling algorithms and their implementation are discussed at the end of this chapter.

3.2. Scheduling Problems

3.2.1. Problems, Algorithms And Complexity

Scheduling problem [6] is the problem of matching elements from different sets, which is formally expressed as a triple (E, S, O), where

• E is the set of examples, each of which is an instance of problem.

• S is the set of feasible solutions for the example.

• O is the object of the problem.

Scheduling problem can be further classified into two categories depending on object O: optimization problem and decision problem. An optimization problem requires finding the best solution among all the feasible solutions in set S. Different from optimization, the aim of decision problem is relatively easy. For a specified feasible solution $s \in S$, problem needs a positive or negative answer to whether the object O is achieved. Clearly, optimization problem is harder than decision problem, because the specified solution only compares with one threshold solution in decision problem, instead of all feasible solutions in optimization problem.

An algorithm is a collection of simple instructions for finding a solution to a problem. It contains three parts: input, method, output. Input is a set of parameters to be dealt with. Method includes describable, controllable, repeatable procedures to realize the aim using input parameters. Output is a result of the problem. Especially for scheduling, the algorithm is a method by which tasks are given access, matched, or allocated to processors. Generally speaking, no absolutely perfect scheduling algorithm exists, because scheduling objectives may conflict with one another. A good scheduler implements a suitable compromise, or applies combination of scheduling algorithms according to different applications. A problem can be solved in seconds, hours or even years depending on the algorithm applied. The efficiency of an algorithm is evaluated by the amount of time necessary to execute it. The running time of an algorithm is stated as a time complexity function relating the inputlength to the number of steps.

There are several kinds of time complexity algorithms that will appear in the following chapters.

• For a constant time algorithm O(1), the maximum amount of running time is bounded by a value that does not rely upon the size of the input.

• For a linear time algorithm O(n), the maximum amount of running time increases linearly with the size of the input.

• For a polynomial time algorithm O (nc) with a constant c, the maximum amount of running time is bounded by a polynomial expression in the size of the input.

• For a exponential time algorithm O (2nc) with a constant c, the maximum amount of running time is bounded by an exponential expression in the size of the input.

If a problem has a polynomial time algorithm, the problem is tractable, feasible, efficient or fast enough to be executed on a computational machine. In computational complexity theory, a complexity class is a set of problems that has the same complexity based on a certain resource [7].

• Class P is the set of decision problems that are solvable in polynomial time on a deterministic Turing machine, which means that a problem of Class P can be decided quickly by a polynomial time algorithm.

• Class NP is the set of decision problems that are solvable in polynomial time on a nondeterministic Turing machine, but a candidate solution of the problem of Class NP can be verified by a polynomial time algorithm, which means that the problem can be verified quickly. Class NP-complete is the set of decision problems, to which all other NP problems can be polynomial transformable, and a NP-complete problem must be in class NP. Generally speaking, NP-complete problems are more difficult than NP problems.

• Class NP-hard is the set of optimization problems, to which all NP problems can be polynomial transformable, but a NP-hard problem is not necessarily in class NP. Although most of NP-complete problems are computationally difficult, some of them are solved with acceptable efficiency. There are some algorithms, the running time of which is not only bounded by the size of input of an example, but also by the maximum number of the examples. These algorithms have pseudopolynomial time complexity. For one problem, if its maximum number is not large, it can be solved quickly. Thus, one NP-complete problem with known pseudo-polynomial time algorithms is called weakly NP-complete, otherwise is called strongly NP-complete, if it can not be solved by a pseudopolynomial time algorithm unless P=NP [7].

3.2.2. Schematic Methods For Scheduling Problem

Scheduling problems belong to a broad class of combinational optimization problems aiming at finding an optimal matching from a finite set of objects, so the set of feasible solutions is usually discrete rather than continuous. An easy problem refers to one with a small number of the examples, so it can be simply worked out by polynomial algorithms or enumerations.

On the contrary a problem is in Class NP-complete if its purpose is making a decision, and is in Class NP-hard if its purpose is optimization. Because an optimization problem is not easier than a decision problem, we only list schematic methods for NP-hard problems. As shown in Figure 3.1, enumeration, heuristic and approximation are three possible solutions, their corresponding algorithms complement each other to give a relatively good answer to a NP-hard problem.

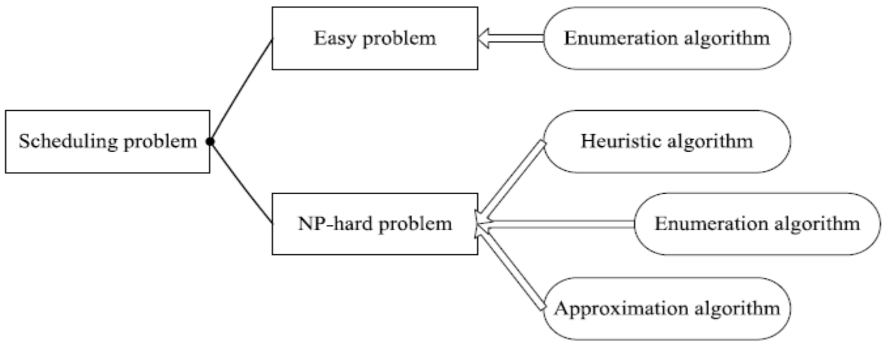


Figure 3.1. Schematic view.

3.2.3. Enumeration Method

For an optimization problem, its optimal solution can be selected if all the possible solutions are enumerated and compared one by one. Exact enumerative algorithms have the exponential time complexity in the worst case. However, for some NP-hard problems in weak sense, when the number in one instance is relatively small, it can be solved by a pseudopolynomial algorithm, the time complexity of which is bounded by a polynomial expression of the input size and the maximum number of the problem. Moreover, there is another kind of enumeration, called implicit enumeration, which evaluates all possible solutions without explicitly listing all of them. Dynamic programming is a practicable implicit enumeration method to solve combinational optimization problems. It divides a problem into a number of stages, and at each stage a decision is required, impacting on the decisions to be made at later stages. The number of stored decisions is exponential to the number of subproblems, so the worst complexity function of dynamic programming algorithms is exponential.

3.2.4. Heuristic Method

Exhaustive enumeration is not feasible for scheduling problems, because only a few special cases of NP-hard problems have exactly-solvable algorithms in polynomial time. For the sake of practice, we tend to find suboptimal solutions that are good enough to balance accuracy and time. Heuristic is a suboptimal algorithm to find reasonably good solutions reasonably fast. It iteratively improves a candidate solution with regard to a given measure of quality, but does not guarantee the best solution. To be more precise, approximation rate rH(e) is introduced to evaluate the accuracy of heuristic algorithms [6].

rH(e) =H(e) OPT(e) (3.1)

where H(e) is the value of the solution constructed by heuristic H for instance e, and OPT(e) is the value of the optimal solution for e. If there is an integer K, all the instances satisfy OPT(e) $\geq$ K, this asymptotic ratio rH can be used to measure the quality of approximation algorithm. The closer rH approaches one, the better the performance is achieved by heuristics. With greedy rules, several common algorithms are shown as follows.

• Next Fit heuristic: The simplest algorithm for bin-packing problem. Each object is assigned to the current bin if it fits, otherwise, it is assigned to a new bin. Approximation rate is rNF $\leq$ 2.

• First Fit heuristic: Each object is assigned to the lowest initialized indexed bin if it fits. A new bin is created only if the new object can not fit any initialized bin. Approximation

rate is rFF $\leq$ 7/4.

• Best Fit heuristic: Each object is assigned to the smallest residual bin if it fits. A new bin is created only if the new object can not fit any initialized bin. Approximation rate is rBF $\leq$ 7/4.

• Next/First/Best Fit Descending heuristic: Objects are first sorted in descending order, and then are ssigned by corresponding heuristics. Approximation rate is rxFD $\leq$ 3/2.

3.2.5. Relaxation Method

Another feasible method to solve NP-hard problems is relaxing some constraints imposed on the original problem. In the new relaxed problem, the solution might be easy to obtain and have a good approximation to that in the original problem. The common relaxation includes:

• Suppose the elements in one instance are all natural numbers, rather than real numbers.

• Suppose the value of one special element remains unchanged, rather than varied.

• Suppose the value of two interrelated elements equal, rather than one being bounded by the other.

• Suppose the value of one element is unit, rather than arbitrary.

• Suppose the type of one element is certain, rather than arbitrary.

More relaxation can be applied without the limit of above presentation.

3.3. Scheduling Hierarchy In Cloud Datacenter

In last section, we introduced related theory about scheduling problems and their schematic methods. From this section, we specify scheduling problems in cloud environments. As a key characteristic of resource management, service scheduling makes cloud computing different from other computing paradigms. Centralized scheduler in cluster system aims at enhancing the overall system performance, while distributed scheduler in grid system aims at enhancing the performance of specific end-users. Compared with them, scheduling in cloud computing is much more complicated. On one hand, centralized scheduler is necessary, because every cloud provider, which promises to provide services to users without reference to the hosted infrastructure, has an individual datacenter. On the other hand, distributed scheduler is also indispensable, because commercial property determines that cloud computing should deal with the QoS requirements of customers distributed worldwide. An important issue of this chapter is to decompose scheduling problems related to cloud computing. Since cloud service is actually a virtual product on a supply chain, the service scheduling can be classified into two basic catagories: user-level and system-level. The hierarchy is shown in Figure 3.2. The user-level scheduling deals with the problem raised by service provision between providers and customers. It mainly refers to economic concerns such as equilibrium of supply and demand, competition among consumers and cost minimization under elastic consumer preference. The system-level scheduling handles resource management within a datacenter. From the point of view of customers, a datacenter is an integration system, which provides uniform services. Actually, the datacenter consists of many physical machines, homogeneous or heterogeneous. After receiving numerous tasks from different users, assigning tasks to physical machines significantly impacts the performance of datacenter. Besides improving the system utilization, some specific requirements should be considered, such as the real-time satisfaction, resource sharing, fault tolerance, etc.

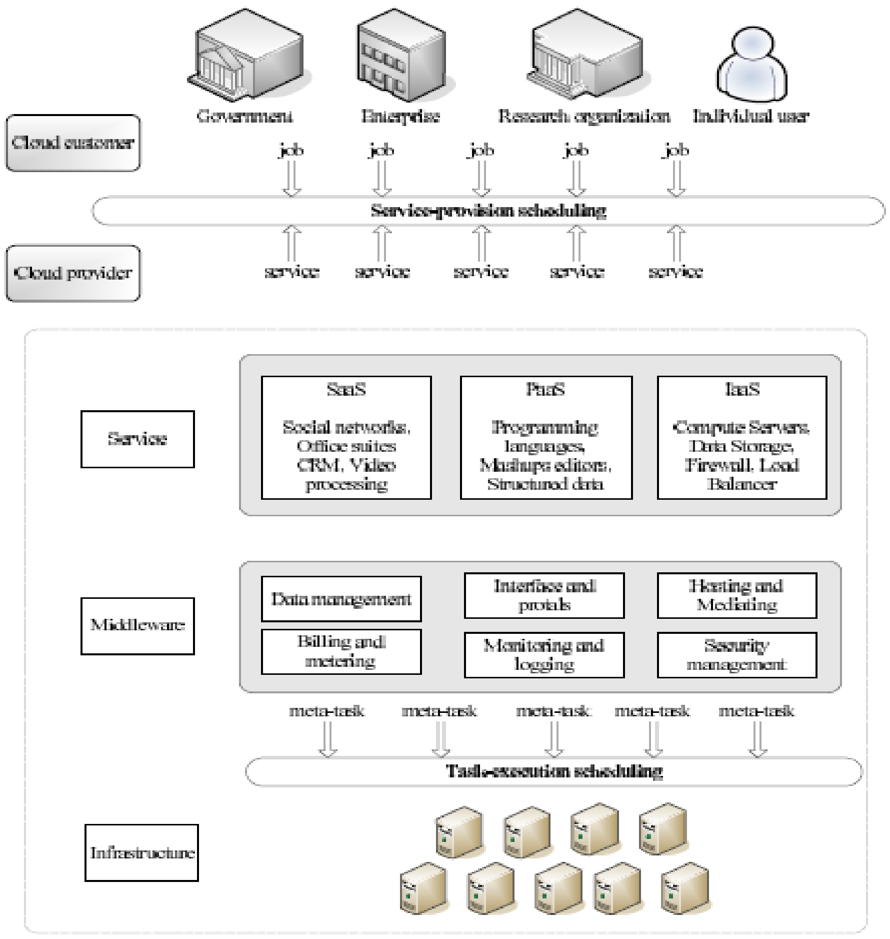


Figure 3.2. Scheduling hierarchy

3.4. Economic Models For Resource-Provision Scheduling

In the past three years, explosion of supply-side cloud service provision has accelerated, cloud solutions become mainstream productions of IT industry. At the same time, these cloud services gradually mature to become more appropriate and attractive to all types of enterprises. The growth of both sides of supply and demand makes the scheduling problems more complex, sophisticated, and even vital in cloud environment. A bad scheduling scheme not only undermines CPU utilization, turnaround time and cumulative throughput, but may also result in terrible consequences, for example providers lose money and even go out of business. Economic models are more suitable for cloud-based scheduling than traditional multiprocessor models, especially for regulating the supply and demand of cloud resources. In economics, market-based and auction-based schedulers handle two main interests. Market-based schedulers are applied when a large number of naive users can not directly control service price in commodity trade. Mainstream cloud providers apply market-based pricing schemes in reality. The concrete schemes vary from provider to provider. As the most successful IaaS provider, Amazon EC2 supports commodity and posted pricing models for the convenience of users. Another alternative is auction-based scheduler, which is adapted to situations where a small number of strategic users seeking to attain a specific service compete with each other. In auctions, users are able to commit the auction price. Amazon spot instance is an example of auction-based model. Instance price adjusts from time to time, depending on the supply and demand. As a result, users should estimate the future price and make its proposal in an auction before placing a spot instance request.

3.4.1. Market-Based Strategies

In cloud service provision, both service providers and users express their requirements through SLAs contracts. Providers need mechanisms that support price specification and increase system utilization, while consumers need schemes that guarantee their objectives are reached. A market-based scheduler aims at regulating the supply and demand for resources. To be specific, the market strategies emphasize the schemes for establishing a service price depending on their customers' requirements. In previous literature, a broker behaving on the behalf of one end-user interacts with service providers to determine a proper price that keeps supply and demand in equilibrium [8].

3.4.1.1. Strategy types

3.4.1.1.1. Commodity model

As a common model in our daily life, service providers specify their service price and charge users according to the amount of resource they consume. Any user is free to choose a proper provider, but has no right to change the service price directly. The amount of their purchase can cause the price to derive from supply and demand. The process of scheduling is executed by brokers. On the behalf of users, each broker identifies several providers to inquire the prices, and then selects one provider which can meet its objective. The consumption of service is recorded and payment is made as agreed.

3.4.1.2. Posted price model

The posted price strategy makes some special offers to increase the market share or to motivate customers to use the service during the off-peak period. The posted price, as a kind of advertisement, has time or usage limitations that are not suitable for all users. Therefore, the scheduling process should be modified in this strategy.

Service providers give the regular price, the cheap offers and the associated conditions of usage. Brokers observe the posted price, and compare whether it can meet the requirement of users. If not, brokers apply commodity strategy as usual. Otherwise, brokers only inquire the provider for availability of posted services, supplementing extra regular service when associated conditions are not satisfied.

3.4.1.3. Bargaining model

In bargaining strategy, price is not given by provider unilaterally, but by both sides of the transaction through bargaining. A prerequisite for bargaining is that the objective functions for providers and brokers must have an intersection, so that they can negotiate with each other as long as their objectives are both met. In this scenario, a broker does not compare all the prices for the same service, but connects with one of the providers directly. The price offered by the provider might be higher than customer expectation, so the broker starts with a very low price, which has the upside potential. The bargaining ends when a mutually agreeable price is reached or when one side is not willing to negotiate any further. In the latter case, broker will connect with other providers and then start bargaining again. Bargaining strategy has an obvious shortcoming, that is, the overhead on communication is very high. The time delay might lead to lower utilization of resources for the provider or shorten deadline of service for the customers. In reality, the number of negotiations can not be infinite, and the bargaining time is always limited.

3.4.1.4. Principles for strategy design

Several market principles should be considered in the process of determining the service price [9].

Equilibrium price refers to a price under which the amount of services bought by buyers is equal to the amount of services produced by sellers. This price tends to be stable unless demand or supply change.

Pareto efficiency describes a situation where no agent can get a better allocation than the initial one without reducing other individual allocations. In other words, resource can not be reallocated in a way that makes everyone better off.

Individual rationality can make price fluctuate around the equilibrium price, which is determined by the process of supply and demand. A higher price provides incentive to produce more resource, so the amount of scarce resource can gradually reach saturation then surplus, and vice-versa. Individual rationality can adjust prices to reach equilibrium instantaneously.

Stability examines whether a scheduling mechanism can be manipulated. Individual agent may not reveal private information truthfully. A stable mechanism allows agents to obtain the best allocation if they submit their truthful information.

Communication efficiency evaluates the communication overhead to capture a desirable global solution. Message passing adds communication overhead on transaction, so additional time is spent on allocation, rather than on computation. A good scheduling mechanism finds out a near-optimum solution efficiently.

3.4.2. Auction Strategies

Unlike in market-based models, an auction-based scheduler is a rule maker, rather than a price maker. The rules include how the users bid for services, how the sale price is determined, who the winning bidder is, how the resource is allocated, whether there are limits on time or proposal price, etc. In auction-based schedulers, price is decided according to the given rules, which benefits consumers by expressing their real requirement strategically, rather than waiting for price ad-justment in a passive manner. Auction-based schedulers are distinguished from each other by several characteristics.

3.4.2.1. Strategy types

3.4.2.1.1. Number of participants

According to different numbers of bidders, auctions are classified into demand auction, supply auction and double auction. English auction is an example of demand auction, in which n buyers bid for one service. This type of auction is the most common form of auction in use today. Dutch auction focuses on demand of suppliers, where m sellers offer the same service for one buyer. Double auction is needed under the condition that the number of buyers and sellers is more than one. In double auction, sellers and buyers both offer bids. The amount of trade is decided by the quantity at which the marginal buy bid is higher than the marginal sell bid. With the growing number of participants, double auction converges to the market equilibrium.

3.4.2.2. Information transparency

Participants in an auction may or may not know the actions of other participants. Both English and Dutch auctions are open auctions, that is, the participants repeatedly bid for the service with the complete information about previous bids of other bidders. Apart from these, there is another type of auction, in which participants post sealed bids and the bidder with highest bid wins. In close auction, bidders can only submit one bid each and no one knows the other bids. Consequently, blind bidders cannot adjust their bids accordingly. Close auction is commonly used for modeling resource provision in multi-agent system, considering the simplicity and effectiveness of the sealed bids.

3.4.2.3. Combinatorial auction

A combinatorial auction is a type of smart market in which participants can place bids on combinations of items, rather than just individual items. Combinatorial auction is appropriate for computational resource auction, where a common procedure accepts bids for a package of items such as CPU cycles, memory, storage, and bandwidth. Combinatorial auctions are processed by bidders repeatedly modifying their proposals until no one increases its bid any more. In each round, auctioneer publishes a tentative outcome to help bidders decide whether increase their bids or not. The tentative outcome is the one that can bring auctioneer the best revenue given the bids. However, finding an allocation of items to maximize the auctioneer’s revenue is NP-complete. A challenge of combinatorial auctions comes from how to efficiently determine the allocation once the bids have been submitted to the auctioneer.

3.4.2.4. Proportion shared auction

In proportion shared auctions, no winner exists, but all bidders share the whole resource with a percentage based on their bids. This type of auction guarantees a maximized utility and ensures fairness among users in resource allocation, which suits limited resource such as time slot, power and spectrum bandwidth [10]. Shares represent relative resource rights that depend on the total number of shares contending for a resource. Client allocations degrade gracefully in overload situations, and clients proportionally benefit from extra resources when some allocations are underutilized.

3.4.2.5. Principles for strategy design

3.4.2.5.1. Game theoretical equilibrium

The auction models applied in cloud service and other computational resource provisioning are listed above, but not limited to these primary types. Generally, auction-based scheduler emphasizes the equilibrium among users rather than supply-demand balance between provider and user. The effectiveness of auction can be analyzed with the help of game theory. Game theory studies multi-person decision making problems. Any player involved in a game makes the best decision, taking into account decisions of the others. A game theoretical equilibrium is a solution, in which no player gains by only changing his own strategy unilaterally. However, this equilibrium does not necessarily mean the best cumulative payoff for all players.

3.4.2.6. Incentive compatibility

In any auction, participants might hide their true preferences. Incentive compatible auction is one in which participants have incentive to reveal their real private information. One bidder can maximize his payoff only if the information is submitted truthfully. One method to realize incentive compatibility is designing a reasonable price payed by auction winner. A good example of incentive compatible auction is Vickery auction. In this sealed price auction, the highest bidder wins, but pays the second highest bid rather than his own. Under this charging rule, biding lower or higher than his true valuation will never increase the best possible outcome.

3.4.3. Economic Schedulers

Economic schedulers have been applied to solve resource management in various computing paradigms, such as cluster, distributed databases, grids, parallel systems, Peer-to-Peer, and cloud computing [11]. Existing middleware applying economic schedulers, not limited to cloud platforms, are introduced. By doing this, we can examine the applicability and suitability of these economic schedulers for supporting cloud resource allocation in practice. This in turn helps us identify possible strengths of these middleware that may be leveraged for cloud environment.

Cluster-on-demand [12] is a service-oriented architecture for networked utility computing. It creates independent virtual clusters for different groups. These virtual clusters are assigned and managed by a cluster broker, supporting tendering and contract-net economic model. The user submits its requirements to all cluster brokers. Every broker proposes a specific contract with the estimated execution time and cost. If the number of brokers proposing contacts is more than one, users then select only one of them as the resource provider. Earning is afforded by users to cluster broker as costs for adhering to the conditions of the contract.

Mosix [13] is a distributed operating system for high performance cluster computing that employs an opportunity cost approach to minimize the overall execution cost of the cluster. It applies commodity model to compute a single marginal cost based on the processor and memory usages of the process. The cluster node with the minimal value of marginal cost is then assigned the process.

Stanford Peers [14]is a peer-to-peer data trading framework, in which both auction and bartering models are applied. A local site wishing to replicate its collection holds an auction to solicit bids from remote sites by first announcing its request for storage space. Each interested remote site then returns a bid, and the site with the lowest bid for maximum benefit is selected by the local site. Besides that, a bartering system supports a cooperative trading environment for producer and consumer participants, so that sites exchange free storage spaces to benefit both themselves and others. Each site minimizes the cost of trading, which is the amount of disk storage space that it has to provide to the remote site for the requested data exchange.

D'Agents [15] is a mobile-agent system for distributed computing. It implements proportion shared auction where agents compete for shared resources. If there is more than one bidder, resources are allocated proportionally. Costs are defined as rates, such as credits per minute to reflect the maximum amount that a user wants to pay for the resource.

Nimrod-G [16] is a tool for automated modeling and execution of parameter sweep applications on Grids. Through broker, the grid users obtain service prices from different resources. Deadline and budget are main constraints specified by the user for running his application. The allocation mechanisms are based on market-based models. Prices of resources thus vary between different executing applications depending on their QoS constraints. A competitive trading environment exists, because users have to compete with one another in order to maximize their own personal benefits.

Faucets [17] is a resource scheduler of computational grid, and its objective is supporting efficient resource allocation for parallel jobs executed on a changing number of allocated processors during runtime on demand. Tendering model is used in Faucets. A QoS contract is agreed before job execution, including payoff at soft deadline, a decreased payoff at hard deadline and penalty after hard deadline. Faucets aims to maximize the profit of resource provider and resource utilization.

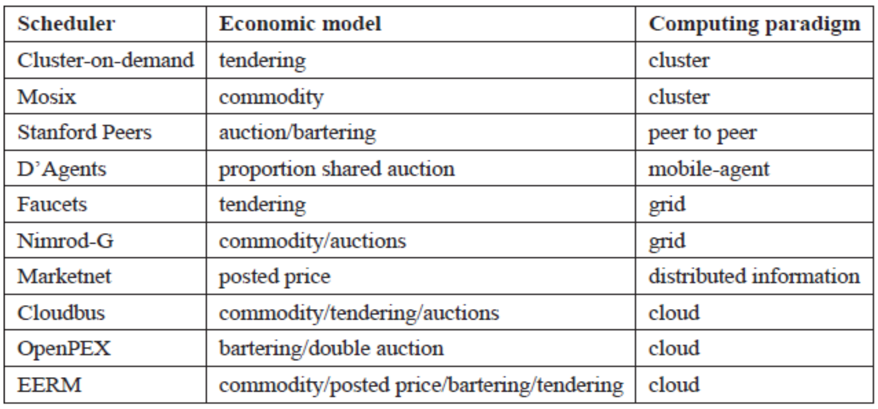
MarketNet [18] is a market-based protection technology for distributed information systems. Posted price model is incorporated. Currency accounts for information usage. MarketNet system advertises resource request by offering prices on a bulletin board. Through observing currency flow, potential intrusion attacks into the information systems are controlled, and the damages are kept to the minimum.

Cloudbus [19] is a toolkit providing market-based resource management strategies to mediate access to distributed physical and virtual resources. A 3rd party cloud broker is built on an architecture that provides a general framework for any other cloud platforms. A number of economic models with commodity, tendering and auction strategies are available for customerdriven service management and computational risk management. The broker supports various application models such as parameter sweep, workflow, parallel and bag of tasks. It has plug-in support for integration with other middleware technologies such as Globus, Aneka, Unicore, etc.

OpenPEX [20] is a resource provisioning system with an advanced reservation approach for allocating virtual resources. A user can reserve any number of instances of virtual machine that have to be started at a specific time and have to last for a specific duration. A bilateral negotiation protocol is incorporated in OpenPEX, allowing users and providers to exchange their offers and counter-offers, so more sophisticated bartering or double auction models are helpful to improve revenue of cloud users.

EERM [21] is a resource broker that enables bidirectional communication between business and resource layers to promote good decision-making in resource management. EERM contains sub-components for performing pricing, accounting, billing, job scheduling, monitoring and dispatching. It uses kinds of market-based mechanisms for allocating network resources. To increase the revenue, overbooking strategy is implemented to mitigate the effects of cancellations and no-shows. A summary of economic schedulers is concluded in Table 3.1.

Table 3.1. Economic schedulers



Scheduler	Economic model	Computing paradigm
Cluster-on-demand	tendering	cluster
Mosix	commodity	cluster
Stanford Peers	auction/bartering	peer to peer
D'Agents	proportion shared auction	mobile-agent
Faucets	tendering	grid
Nimrod-G	commodity/auctions	grid
Marketnet	posted price	distributed information
Cloudbus	commodity/tendering/auctions	cloud
OpenPEX	bartering/double auction	cloud
EERM	commodity/posted price/bartering/tendering	cloud

3.5. Heuristic Models For Task-Execution Scheduling

In cloud computing, a typical datacenter consists of commodity machines connected by highspeed links. This environment is well suited for the computation of large, diverse group of tasks. Tasks belonging to different users are no longer distinguished one fromanother. Scheduling problem in such a context turns out to be matching multi tasks to multi machines. As mentioned in the former section, the optimal matching is an optimization problem, generally with NP-complete complexity. Heuristic is often applied as a suboptimal algorithm to obtain relatively good solutions. This section intensively researches two types of strategies, static and dynamic heuristics. Static heuristic is suitable for the situation where the complete set of tasks is known prior to execution, while dynamic heuristic performs the scheduling when a task arrives. Before further explanation, several preliminary terms should be defined.

• ti: task i

• mj : machine j

• ci: the time when task ti comes

• aj : the time when machine mj is available

• eij : the execution time for ti is executed on mj

• cij : the time when the execution of ti is finished on mj , cij = aj + eij

• makespan: the maximum value of cij , which means the whole execution time. The aim of heuristics is to minimize makespan, that is to say, scheduling should finish execution of metatask as soon as possible.

3.5.1 Static strategies

Static strategies are performed under two assumptions. The first is that tasks arrive simultaneously ci = 0. The second is that machine available time aj is updated after each task is scheduled.

OLB (Opportunistic Load Balancing) schedules every task, in arbitrary order, to next available machine. Its implementation is quite easy, because it does not need extra calculation. The goal of OLB is simply keeping all machines as busy as possible.

MET (Minimum Execution Time) schedules every task, in arbitrary order, to the machine which has the minimum execution time for this task. MET is also very simple, giving the best machine to each task, but it ignores the availability of machines. MET jeopardizes the load balance across machines.

MCT (Minimum Completion Time) schedules every task, in arbitrary order, to the machine which has the minimum completion time for this task. However, in this heuristic, not all tasks can be given the minimum execution time.

Min-min begins with the set T of all unscheduled tasks. Then, the matrix for minimum completion time for each task in set T is calculated. Task with overall minimum completion time is scheduled to its corresponding machine. Next, the scheduled task is removed from T. The process repeats until all tasks are scheduled.

Min-max is similar to Min-min heuristic. Min-max also begins with the set T of all unscheduled tasks, and then calculates the matrix for minimum completion time for each task in set T. Different from min-min, task with overall maximum completion time is selected and scheduled to its corresponding machine. Next, the scheduled task is removed from T. The process repeats until all tasks are scheduled.

GA (Genetic Algorithm) is a heuristic to search for a near-optimal solution in large solution spaces [50]. The first step is randomly initializing a population of chromosomes (possible scheduling) for a given task. Each chromosome has a fitness value (makespan) that results from the scheduling of tasks to machines within that chromosome. After the generation of the initial population, all chromosomes in the population are evaluated based on their fitness value, with a smaller makespan being a better mapping. Selection scheme probabilistically duplicates some chromosomes and deletes others, where better mappings have a higher probability of being duplicated in the next generation. The population size is constant in all generations. Next, the crossover operation selects a random pair of chromosomes and chooses a random point in the first chromosome. Crossover exchanges machine assignments between corresponding tasks. Mutation operation is performed after crossover. Mutation randomly selects a chromosome, then randomly selects a task within the chromosome, and randomly reassigns it to a new machine. After evaluating the new population, another iteration of GA starts, including selection, crossover, mutation and evaluation. Only when stopping criteria are met, the iteration will stop.

SA (Simulated Annealing) uses a procedure that probabilistically allows poorer solutions to be accepted to obtain a better search of the solution space. This probability is based on a system temperature that decreases for each iteration, which implies that a poorer solution is difficulty to be accepted. The initial system temperature is the makespan of the initial scheduling, which is mutated in the same manner as the GA. The new makespan is evaluated at the end of each iteration. A worse makespan might be accepted based on a probability, so the SA finds poorer solutions than Min-min and GA.

Tabu search keeps track of the regions of the solution space which have already been searched so as not to repeat a search near these areas. A scheduling solution uses the same representation as a chromosome in the GA approach. To manipulate the current solution and to move through the solution space, a short hop is performed. The intuitive purpose of a short hop is to find the nearest local minimum solution within the solution space. When the short hop procedure ends, the final scheduling from the local solution space search is added to the tabu list. Next, a new random scheduling is generated, to perform a long hop to enter a new unsearched region of the solution space. After each successful long hop, the short hop procedure is repeated. After the stopping criterion is satisfied, the best scheduling from the tabu list is the final answer.

A∗ is a tree-based search heuristic beginning at a root node that is a null solution. As the tree grows, nodes represent partial scheduling (a subset of tasks is assigned to machines), and leaves represent final scheduling (all tasks are assigned to machines). The partial solution of a child node has one more task scheduled than the parent node. Each parent node can be replaced by its children. To keep execution time of the heuristic tractable, there is a pruning process to limit the maximum number of active nodes in the tree at any one time. If the tree is not pruned, this method is equivalent to an exhaustive search. This process continues until a leaf (complete scheduling) is reached. The listed heuristics above are fit for different scheduling scenarios. The variation of scenarios is caused by the task heterogeneity, machine heterogeneity and machine inconsistence. The machines are consistent if machine mi executes any task faster than machine mj , it executes all tasks faster than mj . These heuristics are evaluated by simulation in article [50]. For consistent machines, GA performs the best, while MET performs the worst. For inconsistent machines, GA and A∗ give the best solution, and OLB gives the worst. Generally, GA, A∗ and min-min can be used as a promising heuristic with short average makespan.

3.5.2. Dynamic strategies

Dynamic heuristics are necessary when task set or machine set is not fixed. For example, not all tasks arrive simultaneously, or some machines go offline at intervals. The dynamic heuristics can be used in two fashions, on-line mode and batch mode. In the former mode, a task is scheduled to a machine as soon as it arrives. In the latter mode, tasks are firstly collected into a set that is examined for scheduling at prescheduled times.

3.5.2.1. On-line mode

In on-line heuristics, each task is scheduled only once, the scheduling result can not be changed. On-line heuristic is suitable for the cases in which arrival rate is low [22].

OLB dynamic heuristic assigns a task to the machine that becomes ready next regardless of the execution time of the task on that machine.

MET dynamic heuristic assigns each task to the machine that performs that task's computation in the least amount of execution time regardless of machine available time.

MCT dynamic heuristic assigns each task to the machine, which results in task's earliest completion time. MCT heuristic is used as a benchmark for the on-line mode [22].

SA (Switching Algorithm) uses theMCT andMET heuristics in a cyclic fashion depending on the load distribution across the machines. MET can choose the best machine for tasks but might assign too many tasks to same machines, while MCT can balance the load, but might not assign tasks machines that have their minimum executing time. If the tasks are arriving in a random mix, it is possible to use the MET at the expense of load balance up to a given threshold and then use the MCT to smooth the load across the machines.

KPB (K-Percent Best) heuristic considers only a subset of machines while scheduling a task. The subset is formed by picking the k best machines based on the execution times for the task. A good value of k schedules a task to a machine only within a subset formed from computationally superior machines. The purpose is to avoid putting the current task onto a machine which might be more suitable for some yet-to-arrive tasks, so it leads to a shorter makespan as compared to the MCT. For all the on-line mode heuristics, KPB outperforms others in most scenarios [22]. The results ofMCT are good, only slightly worse than KPB, owing to the lack of prediction for task heterogeneity.

3.5.2.2. Batch mode

In batch mode, tasks are scheduled only at some predefined moments. This enables batch heuristics to know about the actual execution times of a larger number of tasks.

Min-min firstly updates the set of arrival tasks and the set of available machines, calculating the corresponding expected completion time for all ready tasks. Next, the task with the minimum earliest completion time is scheduled and then removed from the task set. Machine available time is updated, and the procedure continues until all tasks are scheduled.

Max-min heuristic differs from the Min-min heuristic where the task with the maximum earliest completion time is determined and then assigned to the corresponding machine. The Max-min performs better than the Min-min heuristic if the number of shorter tasks is larger than that of longer tasks.

Sufferage heuristic assigns a machine to a task that would suffer most if that particular machine was not assigned to it. In every scheduling event, a sufferage value is calculated, which is the difference between the first and the second earliest completion time. For task tk, if the best machine mj with the earliest completion time is available, tk is assigned to mj . Otherwise, the heuristic compares the sufferage value of tk and ti, the task already assigned to mj . If the sufferage value of tk is bigger, ti is unassigned and added back to the task set. Each task in set is considered only once. Generally, Sufferage gives the smallest makespan among batch mode heuristics [22]. The batch mode performs better than the on-line mode with high task arrival rate.

3.5.3. Heuristic Schedulers

One advantage of cloud computing is that tasks which might be difficult, time consuming, or expensive for an individual user can be efficiently accomplished in datacenter. Datacenter in clouds supports functional separation between the processing power and data storage, both of which locate in large number of remote devices. Hence, scheduling becomes more complicated and challenging than ever before. Since scheduler is only a basic component for the whole infrastructure, no general scheduler can fit for all cloud architectures. In this section, we mainly discuss schedulers used for data-intensive distributed applications.

3.5.3.1. Hadoop

MapReduce is a popular computation framework for processing large-scaled data in mainstream public and private clouds, and it is considered as an indispensable cornerstone for cloud implementation. Hadoop is the most widespread MapReduce implementation for educational or production uses. It enables applications to work with thousands of nodes and petabytes of data. A multi-node Hadoop cluster contains two layers. The bottom is Hadoop Distributed File System (HDFS), which provides data location awareness for effective scheduling of work. Above the file systems is the MapReduce engine, which includes one job tracker and several task trackers. Every tracker inhabits an individual node. Clients submit MapReduce jobs to job tracker, then job tracker pushes work out to available Task Tracker nodes in the cluster [23]. Hadoop is designed for large batch jobs. The default scheduler uses FIFO heuristic to schedule jobs from a work queue. Alternative job schedulers are fair scheduler, capacity scheduler and delay scheduler.

FIFO scheduler [23] applies first in first out heuristic. When a new job is submitted, scheduler puts it in the queue according to its arrival time. The earliest job on the waiting list is always executed first. The advantages are that the implementation is quite easy and that the overhead is minimal. However, throughput of FIFO scheduler is low, since tasks with long execution time can seize the machines.

Fair scheduler [24] assigns equal share of resources to all jobs. When new jobs are submitted, tasks slots that free up are shared, so that each job gets roughly the same amount of CPU time. Fair scheduler supports job priorities as weights to determine the fraction of total compute time that each job should get. It also allows a cluster to be shared among a number of users. Each user is given a separate pool by default, so that everyone gets the same share of the cluster no matter how many jobs are submitted. Within each pool, fair sharing is used to share capacity between the running jobs. In addition, guaranteed minimum share is allowed. When a pool contains jobs, it gets at least its minimum share, but when the pool does not need its full guaranteed share, the excess is split among other running jobs.

Capacity scheduler [25] allocates cluster capacity to multiple queues, each of which contains a fraction of capacity. Each job is submitted to a queue, all jobs submitted to the same queue will have access to the capacity allocated to the queue. Queues enforce limits on the percentage of resources allocated to a user at any given time, so no user monopolizes the resource. Queues optionally support job priorities. Within a queue, jobs with high priority will have access to resources preferentially. However, once a job is running, it will not be preempted for a higher priority job.

Delay scheduler [26] addresses conflict between scheduling fairness and data locality. It temporarily relaxes fairness to improve locality by asking jobs to wait for a scheduling opportunity on a node with local data. When the job that should be scheduled next according to fairness cannot launch a local task, it waits for a short length of time, letting other jobs launch tasks instead. However, if a job has been skipped long enough, it is allowed to launch non-local tasks to avoid starvation. Delay scheduler is effective if most tasks are short compared to jobs and if there are many slots per node.

3.5.3.2. Dryad

Dryad [27] is a distributed execution engine for general data parallel applications, and it seems to be Microsoft’s programming framework, providing similar functionality as Hadoop. Dryad applies directed acyclic graph (DAG) to model applications.

Quincy [28] scheduler tackles the conflict between locality and scheduling in Dryad framework. It represents the scheduling problem as an optimization problem. Min-cost flow makes a scheduling decision, matching tasks and nodes. The basic idea is killing some of the running tasks and then launching new tasks to place the cluster in the configuration returned by the flow solver.

3.5.3.3. Others

To sum up the heuristic schedulers for cloud computing, scheduling in clouds are all about resource allocation, rather than job delegation in HPC or grid computing. However, the traditional meta-schedulers can be evolved to adapt cloud architectures and implementations, considering the development of virtualization technologies. Next, we take several representatives for example as follows

Oracle Grid Engine [29] is an open source batch-queuing system. It is responsible for scheduling remote execution of large numbers of standalone, parallel or interactive user jobs and managing the allocation of distributed resources. Now it is integrated by Hadoop and Amazon EC2, and works as a virtual machine scheduler for Nimbus in cloud computing environment.

Maui Cluster Scheduler [30] is an open source job scheduler for clusters and supercomputers, which is capable of supporting an array of scheduling policies, dynamic priorities, extensive reservations, and fair share capabilities. Now it has developed new features including virtual private clusters, basic trigger support, graphical administration tools, and a Web-based user portal in Moab.

Condor [31] is an open source high-throughput computing software framework to manage workload on a dedicated cluster of computers. Condor-G is developed, provisioning virtual machines on EC2 through the VM Universe. It also supports launching Hadoop MapReduce jobs in Condor’s parallel universe.

gLite [32] is a middleware stack for grid computing initially used in scientific experiments. It provides a framework for building grid applications, tapping into the power of distributed computing and storage resources across the Internet, which can be compared to corresponding cloud services such as Amazon EC2 and S3. Since technologies such as REST, HTTP, hardware virtualization and BitTorrent displaced existing accesses to grid resources, gLite federates both resources from academic organizations as well as commercial providers to keep being pervasive and cost effective.

3.6. Real-Time Scheduling For Cloud Computing

There are emerging classes of applications that can benefit from increasing timing guarantee of cloud services. These mission critical applications typically have deadline requirements, and any delay is considered as failure for the whole deployment. For instance, traffic control centers periodically collect the state of roads by sensor devices. Database updates recent information before next data reports are submitted. If anyone consults the control center about traffic problems, a real-time decision should be responded to help operators choose appropriate control actions. Besides, current service level agreements can not provide cloud users real-time control over the timing behavior of the applications, so more flexible, transparent and trust-worthy service agreement between cloud providers and users is needed in future. Given the above analysis, the ability to satisfy timing constraints of such real-time applications plays a significant role in cloud environment. However, the existing cloud schedulers are not perfectly suitable for real-time tasks, because they lack strict requirement of hard deadlines. A real-time scheduler must ensure that processes meet deadlines, regardless of system load or makespan. Priority is applied to the scheduling of these periodic tasks with deadlines. Every task in priority scheduling is given a priority through some policy, so that scheduler assigns tasks to resources according to priorities. Based on the policy for assigning priority, real-time scheduling is classified into two types: fixed priority strategy and dynamic priority strategy.

3.6.1. Fixed Priority Strategies

A real-time task τi contains a series of instances. Fixed priority scheduling is that all instances of one task have the same priority. The most influential algorithm for priority assignment is Rate Monotonic (RM) algorithm proposed by Liu [33]. In RM algorithm, the priority of one task depends on its releasing rate. The higher the rate is, the higher the priority is. Period Ti is the length of time between two successive instances, and computation time Ci is the time spent on task execution. Since the releasing rate is inverse to its period, Ti is usually the direct criterion to determine task priority. Schedulbility test is to determine whether temporal constraints of tasks can be met at runtime. Exact tests are ideal but intractable, because the complexity of exact tests is NP-hard for non-trivial computational models [34]. Sufficient tests are less complex but more pessimistic. Schedulbility analysis is suitable for the systems whose tasks are known a priori. Sufficient test can be executed by checking whether a sufficient utilization-based condition is met. For example, Liu [33] proved that a set of n periodic tasks using RM algorithm is schedulable if $\sum \frac{C_i}{T_i} \leq n(2^{1/n} - 1)$. The bound is tight in the sense that there are some task sets unschedulable with the utilization that is arbitrarily higher than n(21/n − 1). Actually, many task sets with utilization higher than this bound can be scheduled. Lehoczky [35] proved that the average schedulable utilization, for large randomly chosen task sets, reaches 0.88, much higher than 0.69 of Liu’s result. The desire for more precise and tractable schedulability test pushes researchers to search high utilization bounds under special assumptions, such as appropriate choice of task periods. Exact test permits higher utilization levels to be guaranteed. One approach to solve this problem is that determining the worst-case response time of a task Ri. Once the longest time between arrival of a task and its subsequent instantiations is known, the test can be checked by comparing the deadline Di and the worst-case response time Ri. The complexity of the test comes from the Ri calculation by recursive equations. $R_i = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$. This equation can be solved iteratively, because only a subset of the task release times in the interval between zero and Ti needs to be examined, observed by Harter, Joseph and Audsley independently [36,37,38].

One relaxation of Liu’s model is that task deadline does not exactly equal its period. Therefore, RM algorithm is not optimal for priority assignment. Instead, Leung proposed Deadline Monotonic (DM) algorithm as the optimal policy for such systems, assigning higher priorities to tasks with shorter deadlines than those with longer deadlines [39]. Under this assumption, Lehoczky [40] proposed two sufficient schedulability tests by restricting Di = kTi, where k is a constant across all tasks. Tindell [41] extended exact test for tasks with arbitrary deadlines. A further relaxation is permitting tasks to have unequal offsets. Since the worst-case situation occurs when all tasks share a common release time, utilization bound for sufficient test and response time for exact test in Liu’s model might be too pessimistic. General offsets still remain a problem to efficiently analyze. Under the assumption of specified offsets, RM and DM are no longer optimal, but Audsley [42] showed the optimal priority assignment can be achieved by examining a polynomial number of priority ordering over the task set. Liu’s model and its further extensions are suitable for single processor scheduling. In distributed systems, multiple processors can be scheduled in two approaches, partitioned and global. The former is that each task is assigned to one processor, which executes all incantations of the task. The latter is that tasks complete for the use of all processors. Partition and global

schemes are incomparable in effectiveness, since the required number of processors is not the same [34].

For partitioned policy, the first challenge is to find the optimal partitioning of tasks among processors, which is a NP-complete problem. Therefore, heuristics are used to find good suboptimal static allocations. The main advantage of heuristic approaches is that they are much faster than optimal algorithms while they deliver fairly good allocations. Dhall [43] proved that RMNext-Fit guarantees schedulability of task sets with utilization bound ofm/(1+21/3). Oh [44] showed that RM First-Fit schedules periodic tasks with total uitilizaiton bounded by m(21/2−1). Later, Lopez [45] lifted a tight bound of (m+1)(21/(m+1)−1) for RM First-Fit scheduling. Andersson [46] showed that system utilization can not be higher than (m + 1)/2

for any combination of processor partitioning and any priority assignment. For global policy, the greatest concern is to find an upper bound λ on the individual utilization for RM global scheduling. The small λ presents high system utilization bound. Andersson [46] proved that system utilization bound is m2/(3m−1) with λ = m/(3m−2). Baruah [47] showed that for λ = 1/3 system utilization of at least m/3 can be guaranteed. With arbitrary large λ, Barker [48] showed that the system utilization bound is (m/2)(1 − λ) + λ.

3.6.2. Dynamic Priority Strategies

Dynamic priority assignment is more efficient than the fixed manner, since it can fully utilized the processor for the most pressing tasks. The priorities change with time, varying from one request to another or even during the same request. The most used algorithms are Earliest Deadline First (EDF) and Least laxity First (LLF) [49]. EDF assigns priorities to tasks inversely proportional to the absolute deadlines of the actives jobs. Liu [33] proved that n periodic tasks can be scheduled using EDF algorithm if and only if

$\sum \frac{C_i}{T_I} \le 1.$. LLF assigns the processor to the active task with the smallest laxity. LLF has a large number of context switches due to laxity changes at runtime. Even though both EDF and LLF are optimal algorithms, EDF is more popular in real-time research because of smaller overhead than LLF. Under EDF, schedulability test can be done by processor demand analysis. Processor demand in an interval [t1, t2] is the amount of processing time g(t1, t2) requested by those tasks that must be completed in [t1, t2]. The tasks can be scheduled if and only if any interval of time the total processor demands g(t1, t2) is less than the available time [t1, t2]. Baruah [50] proved that a set of periodic tasks with the same offset can be scheduled if and only if U < 1

and $\forall L > 0, \sum_{i=1}^{n} \left\lfloor \frac{L+T_i-D_i}{T_i} \right\rfloor C_i \le L.$ The sufficient test of EDF is of O(n) complexity if deadline equals period. Otherwise, exact test can be finished in pseudo-polynomial time complexity, when deadline is no longer than period [34]. The research on real-time scheduling is not limited to the issues discussed above. For practicable usage, assumptions can be released, so that researches are extended in a number of ways.

• Not all the tasks have periodic release. Aperiodic server is introduced to permit aperiodic tasks to be accommodated in the periodic models.

• Tasks have resource or precedence relationships. Tasks can be linked by a linear precedence constraint, and communicating via shared resources is allowed to realize task interaction.

• Computation time of tasks varies widely. Some reduced-but-acceptable level of service should be provided when workload exceeds normal expectations.

• Soft real-time applications exist. Control mechanisms can optimize the performance of the systems, and analytic methods are developed to predict the system performance.

3.6.3. Real-Time Schedulers

A scheduler is called dynamic if it makes scheduling decisions at run time, selecting one out of the current set of ready tasks. A scheduler is called static (pre-run-time) if it makes scheduling decisions at compile time. A static scheduler generates a dispatching table for the run-time dispatcher off-line. Generally, real-time schedulers are embedded in corresponding kernels with respect to their scheduling approaches. MARS kernel [51] targets on hard real-time systems for peak load conditions. Fixed scheduling approach is adopted. Schedule is completely calculated offline and is given to the nodes as part of system initialization. All inter-process communications and resource requests are included in the schedule. Nodes may change schedules simultaneously to another pre-calculated schedule. Arts kernel [52] aims at providing a predictable, analyzable, and reliable distributed computing system. It uses the RM/EDF/LLF algorithms to analyze and guarantee hard real-time processes offline. Non-periodic hard real-time processes are scheduled using execution time reserved by a deferrable server. All other processes are scheduled dynamically using a valuefunction scheme. With the augmentation of real-time services, real-time kernel are widely required in cloud computing. However, many kernels are not very capable of satisfying real-time systems requirements, particularly in the multicore context. One solution is applying loadable real-time scheduler as plug-ins into operation systems regardless of kernel configurations. As a result, variant scheduling algorithms are easily installed. A good example is RESCH for Linux kernel, which implements four scheduler plugins with partitioned, semi-partitioned, and global scheduling algorithms [53]. When schedulers step into cloud environment, virtualization is an especially powerful tool. Virtual machines can schedule real-time applications [54], because they allow for a platformindependent software development and provide isolation among applications. For example, Xen provides simplest EDF scheduler to enforce temporal isolation among the different VMs. OpenVMS, a multi-user multiprocessing virtual memory-based operating system, is also designed for real-time applications.

4. CONCLUSIONS

In this chapter, we firstly review the scheduling problems in a general fashion. Then we describe the cloud service scheduling hierarchy. The upper layer deals with scheduling problems raised by economic concerns, such as equilibrium in service providers and consumers, the competition among consumers needing the same service. Market-based and auction models are effective tools, both of which are explained with details and design principles. After that several middleware leveraging these economic models for cloud environment are presented. The lower layer refers to metadata scheduling inside of datacenter. Tasks belonging to different users are no longer distinguished from each other. Scheduling problem is to match multi tasks to multi machines, which can be solved by heuristics. Heuristics are classified into two types. Static heuristic is suitable for the situation where the complete set of tasks is known prior to execution, while dynamic heuristic performs the scheduling when tasks arrive. In cloud-related frameworks such as Hadoop and Dryad, batch-mode dynamic heuristics are most used, and more practical schedulers are developed for special usage. Other meta-schedulers in HPC or grid computing are evolved to adapt cloud architectures and implementations. For commercial purpose, cloud services heavily emphasize time guarantee. The ability to satisfy timing constraints of such real-time applications plays a significant role in cloud environment. We then examine the particular scheduling algorithms for real-time tasks, that is, priority-based strategies. These strategies, already used in traditional real-time kernels, are not very capable of satisfying cloud systems requirements. New technologies, such as loadable real-time plug-ins and virtual machines, are introduced as promising solutions for real-time cloud schedulers.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. Communications of the ACM, 53(4):50–58, 2010.

[2] Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared. In Proceedings of Grid Computing Environments Workshop, pages 1–10, 2008.

[3] D. M. S. Daryl C. Plummer, David W. Cearley. Cloud computing confusion leads to opportunity. Technical report, Gartner Research, 2008.

[4] P. Mell and T. Grance. The NIST Definition of Cloud Computing (Draft). National Institute of Standards and Technology, 53:7, 2010.

[5] M. Armbrust, A. Fox, and R. Griffith. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[6] J. Blazewicz. Scheduling in Computer and Manufacturing Systems. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.

[7] M. Sipser. Introduction to the Theory of Computation. International Thomson Publishing, 1st edition, 1996.

[8] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic models for resource management and scheduling in grid computing. Concurrency and Computation: Practice and Experience, 14:1507–1542, 2002.

[9] Y. Sun, S. Tilak, R. K. Thulasiram, and K. Chiu. Markets, Mechanisms, Games, and Their Implications in Grids, chapter 2, pages 29–48. John Wiley & Sons, Inc., 2009.

[10] Y. kwong Kwok, S. Song, and K. Hwang. Selfish grid computing: Game-theoretic modeling and nash performance results. In Proceedings of International Symposium on Cluster Computing and the Grid, pages 9-12, 2005.

[11] R. Buyya, C. Yeoa, S. Venugopala, J. Broberg, and I. Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems, 25(6):599-616, 2009.

[12] Cluster-on-demand. http://www.cs.duke.edu/nicl/cod/.

[13] Mosix. http://www.mosix.cs.huji.ac.il/.

[14] Stanford peers. http://infolab.stanford.edu/peers/.

[15] D'agents. http://agent.cs.dartmouth.edu/.

[16] D. Abramson, R. Buyya, and J. Giddy. A computational economy for grid computing and its implementation in the nimrod-g resource broker. Future Generation Computer Systems, 18(8):1061–1074, 2002.

[17] L. V. Kale, S. Kumar, M. Potnuru, J. DeSouza, and S. Bandhakavi. Faucets: Efficient resource allocation on the computational grid. In Proceedings of the 2004 International Conference on Parallel Processing, pages 396–405. IEEE Computer Society, 2004.

[18] Dailianas, Y. Yemini, D. Florissi, and H. Huang. Marketnet: Market-based protection of network systems and services - an application to snmp protection. In Proceedings of 19th IEEE International Conference on Computer Communications, pages 1391–1400, 2000.

[19] R. Buyya, S. Pandey, and C. Vecchiola. Cloudbus toolkit for market-oriented cloud computing. In Proceedings of the 1st International Conference on Cloud Computing, pages 24-44. Springer- Verlag, 2009.

[20] S. Venugopal, J. Broberg, and R. Buyya. Openpex: An open provisioning and execution system for virtual machines. Technical Report CLOUDS-TR-2009-8, CLOUDS Laboratory, The University of Melbourne, Australia,, 2009.

[21] E. Elmroth and J. Tordsson. A grid resource broker supporting advance reservations and benchmark-based resource selection. In Lecture Notes in Computer Science, pages 1061–1070. Springer-Verlag, 2005.

[22] M. M. Shoukat, M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. Journal of Parallel and Distributed Computing, 59:107–131, 1999.

[23] D. Borthakur. The Hadoop Distributed File System: Architecture and Design. The Apache Software Foundation, 2007.

[24] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica. Improving mapreduce performance in heterogeneous environments. In R. Draves and R. van Renesse, editors,

Proceedings of Symposium on Operating Systems Design and Implementation, pages 29–42. USENIX Association, 2008.
[25] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Job scheduling for multi-user mapreduce clusters. Technical Report UCB/EECS-2009-55, EECS Department, University of California, Berkeley, Apr 2009.
[26] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In Proceedings of International Conference on EuroSys, pages 265–278, 2010.
[27] Dryad. http://research.microsoft.com/en-us/projects/dryad/.
[28] M. Isard, V. Prabhakaran, J. Currey, U.Wieder, K. Talwar, and A. Goldberg. Quincy: fair scheduling for distributed computing clusters. In Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, pages 261–276. ACM, 2009.
[29] Oracle grid engine. http://www.sun.com/software/sge/.
[30] Maui cluster scheduler. http://www.cluster.com/maui-cluster-scheduler.php.
[31] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the condor experience: Research articles. Journal of Concurrency: Practice and Experience, 17:323-356, February 2005.
[32] C. Ragusa, F. Longo, and A. Puliafito. Experiencing with the cloud over glite. In Proceedings of ICSE Workshop on Software Engineering Challenges of Cloud Computing, pages 53–60. IEEE Computer Society, 2009.
[33] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the Association for Computing Machinery, 20(1):46–61, 1973.
[34] L. Sha, T. Abdelzaher, K.-E. Arzen, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok. Real time scheduling theory: A historical perspective. Real-Time Systems, 28:101-155, November 2004.
[35] J. P. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In Proceedings of IEEE Real-Time Systems Symposium, pages 166–171, 1989.
[36] P. K. Harter, Jr. Response times in level-structured systems. ACM Transaction on Computer Systems, 5:232–248, August 1987.
[37] M. Joseph and P. K. Pandya. Finding response times in a real-time system. The Computer Journal, 29:390–395, 1986.
[38] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. Software Engineering Journal, 8:284-292, 1993.
[39] J. Y. T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. Performance Evaluation, 2:237–250, 1982.
[40] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In Proceedings of the 11th Real-Time Systems Symposium, pages 201–209, 1990.
[41] S. R. Thuel and J. P. Lehoczky. Algorithms for scheduling hard aperiodic tasks in fixed-priority systems using slack stealing. In Proceedings of IEEE Real-Time Systems Symposium, pages 22–33, 1994.
[42] N. C. Audsley, A. Burns, R. I. Davis, K. Tindell, and A. J. Wellings. Fixed priority pre-emptive scheduling: An historical perspective. Real-Time Systems, 8(2-3):173–198, 1995.
[43] C. L. L. S. K. Dhall. On a real-time scheduling problem. Operations Research, 26(1):127–140, 1978.
[44] D.-I. Oh and T. P. Bakker. Utilization bounds for n-processor rate monotonescheduling with static processor assignment. Real-Time Systems, 15(2):183–192, 1998.
[45] J. M. L´opez, J. L. D´ıaz, and D. F. Garc´ıa. Minimum and maximum utilization bounds for multiprocessor rate monotonic scheduling. IEEE Transactions Parallel Distributed Systems, 15(7):642–653, 2004.

[46] B. Andersson, S. Baruah, and J. Jonsson. Static-priority scheduling on multiprocessors. In Proceedings of the 22nd IEEE Real-Time Systems Symposium, page 93. IEEE Computer Society, 2001.
[47] S. K. Baruah and J. Goossens. Rate-monotonic scheduling on uniform multiprocessors. IEEE Transaction on Computers, 52:966–970, July 2003.
[48] T. P. Baker. An analysis of edf schedulability on a multiprocessor. IEEE Transactions on Parallel and Distributed Systems, 16:760–768, 2005.
[49] P. Uthaisombut. Generalization of edf and llf: Identifying all optimal online algorithms for minimizing maximum lateness. Algorithmica, 50:312–328, 2008.
[50] T. D. Braun, H. J. Siegel, N. Beck, L. L. B¨ol¨oni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. Journal of Parallel and Distributed Computing, 61:810–837, June 2001.
[51] J. M. Hyman, J. M. Hyman, A. A. Lazar, A. A. Lazar, G. Pacifici, and G. Pacifici. Real-time scheduling with quality of service constraints. IEEE Journal on Selected Areas in Communications, 9:1052–1063, 1991.
[52] H. Tokuda and C. W. Mercer. Arts: a distributed real-time kernel. SIGOPS Operating Systems Review, 23:29–53, July 1989.
[53] S. Kato, R. Rajkumar, and Y. Ishikawa. A loadable real-time scheduler suite for multicore platform. Technical Report 12, Carnegie Mellon University, Department of Electrical and Computer Engineering, December, 2009.
[54] R. Buyya, S. Pandey, and C. Vecchiola. Cloudbus toolkit for market-oriented cloud computing. In Proceedings of the 1st International Conference on Cloud Computing, pages 24–44. Springer- Verlag, 2009.