Resource Allocation and Consolidation in a Multi-Core Server Cluster Using a Markov Decision Process Model

Yanzhi Wang, Shuang Chen, Hadi Goudarzi, Massoud Pedram
University of Southern California, Los Angeles, CA USA
E-mail: {yanzhiwa, shuangc, hgoudarz, pedram}@usc.edu

Abstract

Distributed computing systems have attracted a lot of attention due to increasing demand for high performance computing and storage. Resource allocation is one of the most important challenges in the distributed systems especially when the clients have some Service Level Agreements (SLAs) and the total profit depends on how the system can meet these SLAs. In this paper, an SLA-based resource allocation problem in a server cluster is considered. The objective is to maximize the total profit, which is the total price gained from serving the clients subtracted by the operation cost of the server cluster. The total price depends on the average request response time for each client as defined in their utility functions, while the operating cost is related to the total energy consumption. A joint optimization framework is proposed, comprised of request dispatching, dynamic voltage and frequency scaling (DVFS) for individual cores, as well as server-level and core-level consolidations. Each core in the cluster is modeled using a continuous-time Markov decision process (CTMDP). A near-optimal hierarchical solution is proposed, consisting of a central manager and distributed local agents. Each local agent employs linear programming-based CTMDP solving method to solve the DVFS problem for the corresponding core. The central manager solves the request dispatching problem and finds the optimal number of turned on cores and servers for request processing, thereby achieving a desirable tradeoff between service request response time and power consumption. Experimental results demonstrate that the proposed near-optimal resource allocation and consolidation algorithm consistently outperforms baseline algorithms.

Keywords

Cloud computing, service level agreement, resource allocation, Markov decision process

1. Introduction

Computing is being transformed to a model consisting of services that are commoditized and delivered in a manner similar to traditional utilities such as water, electricity, gas, and telephony [1]. In such a model, users access services based on their requirements without regard to where the services are hosted or how they are delivered. The recently proposed *Cloud computing* idea [1][2] transforms the infrastructure to a "Cloud" from which businesses and users are able to access applications from anywhere in the world on demand. The computing world is rapidly transforming towards developing software for millions to consume as a service, rather than to run on their individual computers.

In cloud computing, the capabilities of business applications are exposed as sophisticated services that can be accessed over a network. Cloud service providers are incentivized by the profits to be made by charging clients for accessing these services. Clients are attracted by the opportunity for reducing or eliminating costs associated with "in-house" provision of these services. It is essential that the clients have guarantees from providers on service delivery. Typically, these are provided through Service Level Agreements (SLAs) brokered between the providers and consumers. The SLAs include computing power, storage space, network bandwidth, availability and security, etc.

The underlying infrastructure of cloud computing consists of data centers and clusters of servers that are monitored and maintained by the cloud service providers. Service providers often end up over-provisioning their resources in these servers in order to meet the clients' SLAs [3]. Such over-provisioning may increase the cost incurred on the servers in terms of both the electrical energy cost and the carbon emission. Therefore, optimal provisioning or allocation of the resources is imperative in order to reduce the cost incurred on the servers as well as the environmental impact. The problem of optimal resource allocation in the cloud computing framework is therefore crucial and has been investigated in [4][5]. The more general problem of resource allocation and management in distributed computing system has been an active research topic in the recent ten years. There is a number of papers discussing the resource allocation problem in grid computing systems [6][7], in the framework of electronic commerce [8], in autonomic computing systems [9][10], and in clusters of servers [11].

In this paper, we consider the problem of SLA-based resource allocation optimization in a cluster of servers. Multiple clients exist in the resource allocation framework, each generating requests in a different rate. Clients in this system are application software that requires processing, data storage, and communication resources. Each client in this system has a pre-defined *utility function* based on its response time requirements. The server cluster consists of multiple potentially heterogeneous servers, each comprised of a number of potentially heterogeneous cores. The total profit in this system is the total price gained from serving the clients subtracted by the cost of operating the turned on

*This research is sponsored in part by a grant from the National Science Foundation.

servers in the system, where the operation cost of turned on servers is proportional to their energy consumptions.

Different from the prior work, we propose a joint optimization framework considering the optimal request dispatching, dynamic voltage and frequency scaling (DVFS) in each core of the server cluster, as well as server-level and core-level consolidations. Each core in the server cluster is modeled using a continuous-time Markov decision process (CTMDP) [17][18], in which actions are execution frequencies (and supply voltages) for processing the service requests. We know that a higher execution frequency will result in a shorter response time, but a significant increase in power consumption [12][13][14].

We propose a near-optimal hierarchical solution of the resource allocation problem consisting of a central resource manager and distributed local agents. Each local agent employs linear programming-based Markov decision process solving method [18] for solving the DVFS problem for each core, judiciously selecting the most appropriate execution frequency based on the number of waiting requests. The local agents are used to parallelize the solution and decrease the decision time, thereby helping with the scalability. The central manager utilizes optimization methods for solving the request dispatching problem based on the optimization results of the distributed agents. It also finds the optimal number of turned on cores and servers for request processing (i.e., performing core-level and server-level consolidations) in order to achieve a desirable balance between the static and dynamic power components in the servers [15][16]. Experimental results demonstrate that the proposed near-optimal resource allocation and consolidation algorithm consistently outperforms baseline algorithms.

The rest of this paper is organized as follows. The system model and problem formulation are presented in Section 2. The optimization problem formulation and solution are provided in Section 3 and 4, respectively. Experimental results are presented in Section 5 and the conclusion is in the last section.

2. System Model and Problem Formulation

2.1. Overall System Modeling

Figure 1 shows the structure of the target resource allocation system with a set of N clients, a server cluster of M potentially heterogeneous servers, as well as a central resource management node. The central manager has information about the server cluster as well as the clients.

Each client in the system is identified by a unique ID, represented by index i. Each server in the cluster is similarly identified by a unique ID, captured by index j. The j-th server in the cluster consists of K_j potentially heterogeneous cores, indexed by k. Service requests generated by a single client can be assigned to multiple cores in more than one servers. The request dispatcher assigns a request from the i-th client to the k-th core in the j-th server with probability p_{ijk}. These probability values are the optimization variables in the resource allocation optimization framework.

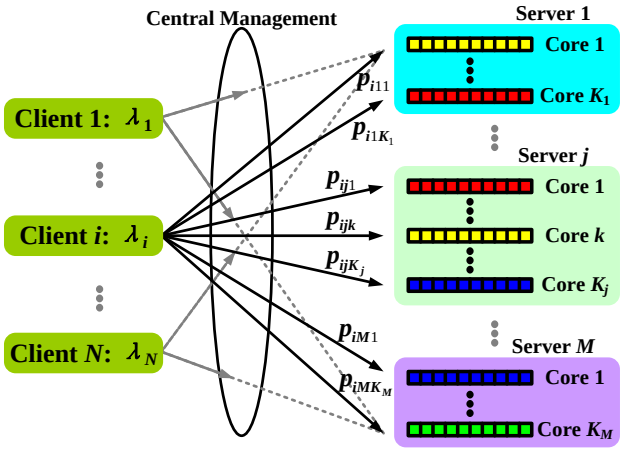

Figure 1: Architecture of the resource allocation problem in the cloud computing system.

In order to find the analytical form of the response time, requests generated from each i-th client are assumed to follow a Poisson process with a mean generating rate of λ_i (predicted based on the behavior of the client.) According to the properties of the Poisson distribution, service requests that are generated from the i-th client and dispatched to the k-th core in the j-th server follow a Poisson process with a mean rate of $p_{ijk} \cdot \lambda_i$ [19]. The average service request arrival rate of that core is given by $\sum_{i=1}^{N} p_{ijk} \cdot \lambda_i$.

2.2. Service Queue Modeling Using CTMDP

We model the service queue (SQ) in each core using CTMDP. CTMDP is a more detailed and accurate model than the Generalized Processor Sharing (GPS) model [20]. A CTMDP is a controllable continuous-time Markov process satisfying the Markovian property [18]. It is comprised of a set of states $\boldsymbol{S}$ and a finite set of actions $\boldsymbol{A}$. The state transition rates are controlled by actions $a \in \boldsymbol{A}$. We are charged by a cost rate function $c(s,a)$ when the system is at state $s \in \boldsymbol{S}$ and we choose action $a \in \boldsymbol{A}$. A *policy* $\pi = \{(s,a)|s \in \boldsymbol{S}, a \in \boldsymbol{A}\}$ is a set of state-action pairs for all states of the CTMDP. We use notation $\pi(s) = a$ to specify the action that is chosen in state s according to the policy π. We consider the class of *stationary* policies. An *optimal policy* is the one that minimizes the total expected cost. The CTMDP optimization problem targets at finding the optimal policy.

Given a CTMDP with n states, its parameterized *generator matrix* $\mathbf{G}(a)$ is defined as an $n \times n$ matrix. Each entry $\sigma_{s,s'}(a)$ in $\mathbf{G}(a)$ is called the *average transition rate* from state s to another state s' when the action a is applied, which is calculated by

$$\sigma_{s,s'}(a) = \frac{\delta_{s,s'}(a)}{\tau_{s,s'}(a)}, \quad \text{for } s \neq s', \tag{1}$$

where $\tau_{s,s'}(a)$ is the average transition time from state s to state s' when action a is applied. $\delta_{s,s'}(a) = 1$ if s' is one of the destination states of the action a, and $\delta_{s,s'}(a) = 0$ otherwise. Interested readers may refer to [18] for more details of the CTMDP.

As illustrated in Figure 2, the SQ in each k-th core in the j-th server ($1 \leq j \leq M, 1 \leq k \leq K_j$) is modeled as a

stationary CTDMP with a state set $\boldsymbol{S} = \{0,1,2,\dots,Q\}$, where Q is the maximum length of the queue. The action set $\boldsymbol{A} = \{f_{min},\dots,f_{max}\}$ corresponds to a set of possible core frequencies while an action $a \in \boldsymbol{A}$ corresponds to a specific execution frequency. When a service request (from any client) is dispatched to this core, the state of the SQ is autonomously incremented by one unless the SQ is full. The average transition rate from state s to state $s+1$ is given by $\sum_{i=1}^{N} p_{ijk} \cdot \lambda_i$, and is independent of the action (i.e., the execution frequency) chosen by the core. When a service request has been serviced by the core, the index of the state of the SQ is autonomously decremented by one. The average transition rate $\mu_{jk}(a)$ in Figure 2 from state s to state $s-1$ is a function of the action (execution frequency) a chosen by the core. A higher execution frequency will result in a shorter average request service time, and thereby, a larger average transition rate $\mu_{jk}(a)$, and vice versa. The average *response time* of a service request in the *k*-th core in the *j*-th server, denoted by $\bar{R}_{jk}$, is the sum of the average waiting time in the SQ and the average request processing time. $\bar{R}_{jk}$ depends on the policy π_{jk} chosen in the core and the probability vector $\boldsymbol{p}_{jk} = (p_{1jk}, p_{2jk}, \dots, p_{Njk})$, denoted by $\bar{R}_{jk}(\pi_{jk}, \boldsymbol{p}_{jk})$.

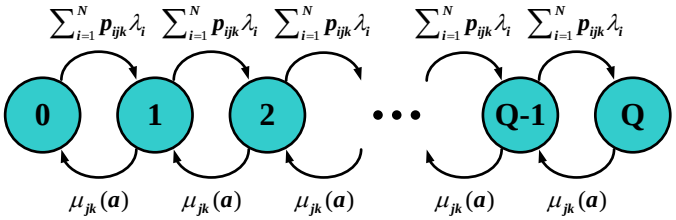

Figure 2: CTMDP model of the SQ in each core.

Power consumption in each core consists of a *dynamic power consumption* part when the core is active (i.e., when it is processing service requests) and a *static power consumption* part as long as the core is ON (i.e., no matter whether the core is active or idle.) The dynamic power consumption $P_{d,jk}^{c}$ is a superlinear function of the core frequency a, denoted by $P_{d,jk}^{c}(a)$ [12][13][14]. In other words, a higher execution frequency will result in a significant increase in its dynamic power consumption. The average dynamic power consumption $\bar{P}_{d,jk}^{c}$ of the *k*-th core in the *j*-th server depends on π_{jk} and $\boldsymbol{p}_{jk}$, denoted by $\bar{P}_{d,jk}^{c}(\pi_{jk}, \boldsymbol{p}_{jk})$. On the other hand, the static power consumption $P_{s,jk}^{c}$ is a constant as long as the *k*-th core in the *j*-th server is turned on by the central manager for processing requests. We may also use $\bar{P}_{s,jk}^{c}$ to denote the static power consumption since it can be viewed as an average value. We capture the effect of power consumption in the cost rate function since the power consumption is directly related to the energy cost. We will provide the details of the cost rate function in each core in Section 4.

The objective of the resource management problem is to maximize the total profit of the server cluster from serving the clients. In this system, decision making intervals can be defined based on the behavior of the dynamic parameters in the system. This is because the solution found by the presented algorithm is acceptable only when the parameters used to find the solution are approximately valid. Although some small changes in the parameters can be effectively tracked and responded to by proper reactions of the central resource manager in the server cluster, large changes cannot be handled in this way. In the remainder of this paper, the resource allocation problem at each decision epoch is presented and a solution is provided, but we do not discuss the estimation, prediction, and dynamic changes in the system because these issues are out of scope of this paper.

3. Optimization Problem Formulation

Let $U_i(R)$ denote the non-increasing utility function of the *i*-th client with the average response time equal to R. Let x_j denote the pseudo-Boolean integer to represent if the *j*-th server is ON ($x_j = 1$) or OFF ($x_j = 0$). As long as the *j*-th server is turned on, it incurs an (average) static power consumption of $\bar{P}_{s,j}^{s}$. Let y_{jk} denote the pseudo-Boolean integer to represent if the *k*-th core in the *j*-th server is ON ($y_{jk} = 1$) or OFF ($y_{jk} = 0$). If $x_j = 0$, we have $y_{jk} = 0$ for all $1 \le k \le K_j$. In the optimization problem, p_{ijk}'s, π_{jk}'s, x_j's, and y_{jk}'s are optimization variables. Other parameters are either constants or functions of these variables.

The overall resource allocation and consolidation problem for the server cluster is formulated as a profit maximization problem as below:

Find the optimal p_{ijk}'s, π_{jk}'s, x_j's, and y_{jk}'s.

Maximize:

$$\sum_{i=1}^{N} \lambda_i \cdot U_i\left(\sum_{j=1}^{M}\sum_{k=1}^{K_j} p_{ijk} \cdot \bar{R}_{jk}(\pi_{jk}, \boldsymbol{p}_{jk})\right) - Price \sum_{j=1}^{M}\left(x_j \bar{P}_{s,j}^{s} + \sum_{k=1}^{K_j} y_{jk}\left(\bar{P}_{s,jk}^{c} + \bar{P}_{d,jk}^{c}(\pi_{jk}, \boldsymbol{p}_{jk})\right)\right) \quad (2)$$

Subject to:

$$x_j \in \{0,1\}, \quad \text{for } \forall j \in \{1,2,\dots,M\}, \quad (3)$$

$$y_{jk} \in \{0,1\}, \text{for } \forall j \in \{1,2,\dots,M\} \text{ and } \forall k \in \{1,\dots,K_j\}, \quad (4)$$

$$0 \le p_{ijk} \le 1, \quad \text{for } \forall i,j,k, \quad (5)$$

$$\sum_{j=1}^{M}\sum_{k=1}^{K_j} p_{ijk} = 1, \quad \text{for } \forall i \in \{1,2,\dots,N\}, \quad (6)$$

$$\sum_{i=1}^{N} p_{ijk} \cdot \lambda_i < \mu_{jk}(f_{max}), \quad \text{for } \forall j,k, \quad (7)$$

$$y_{jk} \ge \frac{\sum_{i=1}^{N} p_{ijk} \cdot \lambda_i}{\mu_{jk}(f_{max})}, \quad \text{for } \forall j,k, \quad (8)$$

$$x_j \ge \frac{\sum_{k=1}^{K_j} y_{jk}}{K_j}, \quad \text{for } \forall j \in \{1,2,\dots,M\}, \quad (9)$$

where *Price* is the *unit energy price* at this decision epoch. Also the constraints in the CTMDP problem formulation for each core should also be satisfied.

In the objective function (2), the first term is the total price gained from serving the service requests, and the

second term is the total energy cost for operating the server cluster. Constraints (3) – (5) specify the domains of variables. Constraint (6) ensures that all requests generated by a client are served. Constraint (7) shows the upper limit on the average service request arrival rate to a core, i.e., it should be smaller than the maximum average service processing rate of that core when it is running at the maximum execution frequency. Constraints (8) and (9) determine the turned on cores and servers, respectively, based on the allocated resources.

The overall resource allocation and consolidation problem is integrated with a set of CTMDP optimization problems (i.e., finding the optimal policy in the CTMDP for each core.) This problem is a mixed integer nonlinear programming problem. The problem cannot be solved using the conventional convex optimization methods since the objective function is neither convex nor concave even if the optimal values of Boolean variables x_j's and y_{jk}'s are given in prior, i.e., the servers and cores that are turned on for service request processing are given in prior.

4. Optimization Methods

The resource allocation and consolidation optimization problem presented in the previous section is a hard problem due to the non-convexity, the existence of Boolean variables, and the integration of a set of CTMDP optimization problems. The simple problem solvers cannot solve this problem except in the case of very small input size by running exhaustive search or by using stochastic optimization methods such as the Simulated Annealing or Genetic Algorithm. In this section, a near-optimal solution is presented for this problem.

We use a linear-form decreasing utility function in the optimization problem, i.e., $U_i(R) = \beta_i - \alpha_i \cdot R$. The objective function (2) becomes:

$$\sum_{i=1}^{N} \lambda_i \cdot \left(\beta_i - \alpha_i \cdot \sum_{j=1}^{M} \sum_{k=1}^{K_j} p_{ijk} \cdot \bar{R}_{jk}(\pi_{jk}, \boldsymbol{p}_{jk}) \right) - Price \sum_{j=1}^{M} \left(x_j \bar{P}^s_{s,j} + \sum_{k=1}^{K_j} y_{jk} \left(\bar{P}^c_{s,jk} + \bar{P}^c_{d,jk}(\pi_{jk}, \boldsymbol{p}_{jk}) \right) \right) \quad (10)$$

The proposed near-optimal solution is based on distributed decision making instead of only centralized management. This is because of the relatively high complexity of the solution when only the centralized management method is employed. On the other hand, distributed decision making can handle the problem in parallel and reduce the time required to reach a good solution by a large factor with limited amount of communication effort. The following observation is the basis of distributed decision making.

Observation I: When p_{ijk}'s, x_j's, and y_{jk}'s are given, maximizing (10) in a centralized way is equivalent to minimizing the following objective at each *k*-th core in the *j*-th server with $x_j = 1$ and $y_{jk} = 1$:

$$\sum_{i=1}^{N} \alpha_i \lambda_i p_{ijk} \bar{R}_{jk}(\pi_{jk}, \boldsymbol{p}_{jk}) + Price \cdot \bar{P}^c_{d,jk}(\pi_{jk}, \boldsymbol{p}_{jk}). \quad (11)$$

In this way, we separate the solution into a centralized resource management algorithm and a set of distributed local agents. Each local agent solves the CTMDP optimization problem (i.e., the DVFS problem) for the corresponding core, finding the optimal policy π_{jk}. The optimal π_{jk} achieves a desirable tradeoff between the average response time and power consumption. The central management algorithm solves the request dispatching problem and finds the optimal p_{ijk} values, which determines the most appropriate server(s) and core(s) for request dispatching. The central management algorithm also performs server-level and core-level consolidations, i.e., finding the optimal x_j's and y_{jk}'s, in order to achieve a desirable balance between the static and dynamic power components in the servers. We elaborate the details in the following three subsections.

4.1. The Local Agents

We consider the local agent of the *k*-th core in the *j*-th server ($x_j = 1$ and $y_{jk} = 1$) when the p_{ijk} values are given. We denote this local agent the (j,k)-agent. The local agent finds the optimal policy π_{jk} in order to minimize the objective function (11). We properly set the cost rate function in the CTMDP such that the objective function (11) is minimized when we solve the CTMDP problem. We have the following observation based on the Little's theorem [23].

Observation II: Suppose that the cost rate function in the CTMDP is $c(s,a) = w_1 \cdot |s| + w_2 \cdot P^c_{d,jk}(a)$ when the system is at state $s \in \boldsymbol{S}$ and we choose action (frequency) $a \in \boldsymbol{A}$, where $|s|$ is the number of requests waiting or being processed in the SQ, and w_1, w_2 are relative weights greater than or equal to zero. We minimize $w_1 \cdot \left(\sum_{i=1}^{N} p_{ijk} \cdot \lambda_i\right) \cdot \bar{R}_{jk}(\pi_{jk}, \boldsymbol{p}_{jk}) + w_2 \cdot \bar{P}^c_{d,jk}(\pi_{jk}, \boldsymbol{p}_{jk})$ when we solve the CTMDP optimization problem.

Based on Observation II, we set the cost rate function to be:

$$c(s,a) = \frac{\sum_{i=1}^{N} \alpha_i \cdot p_{ijk} \cdot \lambda_i}{\sum_{i=1}^{N} p_{ijk} \cdot \lambda_i} \cdot |s| + Price \cdot P^c_{d,jk}(a). \quad (12)$$

This cost rate function will result in an optimal tradeoff between the average response time and power consumption.

The CTMDP optimization problem is formulated as a linear programming problem as follows:

$$\min_{\{f_s(a)\}} \left(\sum_s \sum_a f_s(a) \cdot \gamma_s(a) \right), \quad (13)$$

where $f_s(a)$ denotes the frequency that state s is entered in and action a is chosen in that state. $\gamma_s(a)$ is the expected cost when the system is in state s and action a is chosen. $\gamma_s(a)$ is calculated as:

$$\gamma_s(a) = \tau_s(a) \cdot c(s,a), \quad (14)$$

where $\tau_s(a) = 1/\sum_{s' \neq s} \sigma_{s,s'}(a)$ is the expected duration of time that the system will stay in state s when action a is chosen, and $\sigma_{s,s'}(a)$ is the average transition rate from state s to state s' when action a is chosen, as defined in (1).

The linear programming problem is solved for variables $f_s(a)$ while satisfying the constraints given below:

$$\sum_{a} f_s(a) = \sum_{s' \neq s} \sum_{a'} f_{s'}(a') \cdot p_{s',s}(a'), \text{ for } \forall s, \quad (15)$$

$$\sum_{s} \sum_{a} f_s(a) \cdot \tau_s(a) = 1, \quad (16)$$

$$f_s(a) \geq 0, \quad \text{for } \forall s \in \boldsymbol{S}, \quad (17)$$

where $p_{s,s'}(a)$ denotes the probability that the system will next come to state s' if it is currently in state s and action a is chosen. Constraints (15) – (17) capture the properties of a CTMDP.

We utilize standard linear programming solver such as the MOSEK [24] to solve the CTMDP optimization problem. We find the optimal policy π_{jk} subsequently, as described in [18].

In order to facilitate the further optimizations, we define an *equivalent M/M/1 queue* for the CTMDP of the *k*-th core in the *j*-th server with policy π_{jk}. The average request processing rate of the equivalent M/M/1 queue is given by

$$\mu_{eq,jk} = \sum_{i=1}^{N} p_{ijk} \cdot \lambda_i + \frac{1}{\bar{R}_{jk}(\pi_{jk}, \boldsymbol{p}_{jk})}, \quad (18)$$

which implies that the equivalent M/M/1 queue has the same average service request response time $\bar{R}_{jk}(\pi_{jk}, \boldsymbol{p}_{jk})$ as the CTMDP when the average request arrival time is $\sum_{i=1}^{N} p_{ijk} \cdot \lambda_i$. Similarly, we assume that the dynamic power consumption of the core is $P^c_{eq,d,jk}$ when processing requests. We derive the $P^c_{eq,d,jk}$ values through intrapolation.

4.2. Optimal Request Dispatching

In the optimal request dispatching problem solved by the central resource manager, we are given the x_j's, y_{jk}'s as well as the optimal policy π_{jk}'s. The objective is to find the optimal p_{ijk} values in order to maximize the objective function (10). The main difficulty in this problem is that $\bar{R}_{jk}(\pi_{jk}, \boldsymbol{p}_{jk})$ and $\bar{P}^c_{d,jk}(\pi_{jk}, \boldsymbol{p}_{jk})$ are implicit functions of p_{ijk}'s. Hence, in the first step we approximate $\bar{R}_{jk}(\pi_{jk}, \boldsymbol{p}_{jk})$ and $\bar{P}^c_{d,jk}(\pi_{jk}, \boldsymbol{p}_{jk})$ using explicit functions of p_{ijk}'s.

We use the equivalent M/M/1 queue defined in Section 4.1 to approximate the CTMDP for each core. The approximate average request response time is $1/(\mu_{eq,jk} - \sum_{i=1}^{N} p_{ijk} \cdot \lambda_i)$ for each core. The approximate average percentage of time that core is active (i.e., having one or more requests waiting or being processed) is $\sum_{i=1}^{N} p_{ijk} \cdot \lambda_i / \mu_{eq,jk}$ [23]. Therefore, the approximate average dynamic power consumption is

$$\frac{\sum_{i=1}^{N} p_{ijk} \cdot \lambda_i}{\mu_{eq,jk}} \cdot P^c_{eq,d,jk}. \quad (19)$$

Then the optimal request dispatch problem becomes:

Minimize:

$$\sum_{j=1}^{M} \sum_{k=1}^{K_j} \frac{\sum_{i=1}^{N} \alpha_i \cdot \lambda_i \cdot p_{ijk}}{\mu_{eq,jk} - \sum_{i=1}^{N} \lambda_i \cdot p_{ijk}} + Price \cdot \sum_{j=1}^{M} \sum_{k=1}^{K_j} \left(\frac{\sum_{i=1}^{N} p_{ijk} \cdot \lambda_i}{\mu_{eq,jk}} \cdot P^c_{eq,d,jk} \right). \quad (20)$$

Subject to the constraints:

$$0 \leq p_{ijk} \leq 1, \quad \text{for } \forall i, j, k, \quad (21)$$

$$\sum_{j=1}^{M} \sum_{k=1}^{K_j} p_{ijk} = 1, \quad \text{for } \forall i \in \{1, 2, \ldots, N\}, \quad (22)$$

$$\sum_{i=1}^{N} p_{ijk} \cdot \lambda_i < \mu_{eq,jk}, \quad \text{for } \forall j, k, \quad (23)$$

$$p_{ijk} = 0 \text{ if } y_{jk} = 0, \quad (24)$$

The optimal request dispatching problem maximizes the sum of a set of linear fractional functions of the variables p_{ijk}'s. Effective algorithms exist in the literature [25] for finding a near-optimal solution of this kind of problem effectively. We exploit the FP algorithm [25] for solving this problem.

We integrate the request dispatching optimization performed by the central resource manager with the local agents, and thereby, we derive an iterative distributed near-optimal solution of the resource allocation problem with given turned-on servers and cores (i.e., with the given x_j's and y_{jk}'s.) Algorithm 1 shows the pseudo-code of the proposed distributed near-optimal solution.

Algorithm 1: Distributed Near-Optimal Solution of the Resource Allocation Problem with Given Turned-on Servers and Cores.

Given x_j's and y_{jk}'s.

Initialize the p_{ijk} values.

Do the following procedure iteratively:

> The central resource manager sends commands to all the local agents with $x_j = 1$ and $y_{jk} = 1$.
>
> **For each** (j,k)-agent with $x_j = 1$ and $y_{jk} = 1$ (all the local agents do the following procedure in parallel):
>
> > Perform CTMDP optimization and find the optimal policy π_{jk} based on the p_{ijk} values.
> >
> > Send response to the central resource manager.
>
> **End**
>
> The central resource manager performs request dispatching optimization and finds the optimal p_{ijk} values based on the obtained optimal π_{jk}'s from local agents.

Until the solution converges.

4.3. Core-Level and Server-Level Consolidations

In this section, we consider the core-level and server-level consolidations performed by the central resource manager. The general idea is that we determine the optimal set of turned on servers and cores in an outer loop while we perform optimal request dispatching and DVFS (i.e., Algorithm 1) in the inner loop. We first turn on all the servers and cores and then turn off a subset of cores or a server in one execution of the outer loop. We terminate execution as long as the calculated total profit of the server cluster from Algorithm 1 stops increasing when we further turn off servers or cores. In this way, we achieve a near-optimal trade-off between the dynamic and static power

consumptions in the server cluster through effective consolidation methods. Algorithm 2 provides the pseudo-code of the proposed consolidation procedure with the combination of optimal request dispatching and DVFS. More advanced algorithms for core-level and server-level consolidations can be developed. However, they are out of the scope of the present paper.

Algorithm 2: Distributed Near-Optimal Solution of the Whole Resource Allocation and Consolidation Problem.

Initialization: Turn on all servers and cores.

Do the following procedure:

> Run Algorithm 1 on the set of turned on servers and cores and calculate the total profit.
>
> Turn off a subset of L cores or a server with the minimum average service request arrival rate(s), and set the corresponding x_j and y_{jk} values to zero.

Until the total profit of the server cluster stops increasing.

5. Experimental Results

In this section, we implement the resource allocation framework and compare the proposed near-optimal resource allocation and consolidation algorithm with baseline resource allocation algorithms.

We consider a server cluster of 3 heterogeneous servers consisting of 6 cores, 6 cores, and 8 cores, respectively. We consider 4 clients in the system. We use normalized amounts of most of the parameters in the system instead of their real values. The average service request generating rate of each client is a uniformly distributed random variable between 2.5 and 4.5 (we will change these parameters later in the experiments.) The minimal average service request processing rate in each core (i.e., when it is running at its minimal execution frequency) is a uniformly distributed parameter between 1.0 and 1.5. Each core can perform at five different execution frequencies (i.e., different actions in the CTMDP-based DVFS setup), which are 1 ×, 1.25 ×, 1.5 ×, 1.75 ×, and 2 × of its minimal execution frequency, respectively. Since the average service request processing rate is proportional to the core's execution frequency, the different average service request processing rates in each core are 1 ×, 1.25 ×, 1.5 ×, 1.75 ×, and 2 × of its minimum average service request processing rate, respectively. The (instantaneous) dynamic power consumption of each core is assumed to be proportional to the square of its execution frequency (or equivalently, proportional to the square of its average service request processing rate.) The (average) static power consumption of each core is assumed to be half as the dynamic power consumption when that core is running at its minimum execution frequency level. The maximum queue length Q is equal to 10. For the utility functions, each α_i value is assumed to be a uniformly distributed random variable between 1 and 1.5, and the β_i values are assumed to be equal to 7. We change the unit energy price $Price$ in the experiments and derive different results on the total profit in the server cluster.

Since there is no previous work addressing the combination of optimal request dispatching, optimal DVFS control for individual cores, and server-level and core-level consolidations, we consider three baseline systems without the CTMDP-based optimization ability for individual cores. In Baseline 1, each core runs at its maximum execution frequency in order to minimize the average service request response time. In Baseline 2, each core runs at its minimum possible execution frequency in order to minimize its dynamic power consumption (of course the average service request processing rate should be higher than the average service request arrival rate.) In Baseline 3, each core runs at its medium execution frequency (i.e., 1.5 × its minimum frequency) in order to achieve a balance between response time and power consumption. All the three baseline systems distribute the service requests from each client with equal probability to all the cores in the server cluster.

Figure 3 illustrates the normalized total profit versus the unit energy price of the proposed near-optimal algorithm and three baseline algorithms. We can observe from Figure 3 that the proposed near-optimal algorithm consistently outperforms the three baseline algorithms. When the unit energy price is 1.0, the total profit obtained by the proposed algorithm is 50.2%, 83.7%, and 25.3% higher than Baseline 1, Baseline 2, and Baseline 3, respectively. When the unit energy price is 1.8 or more, the total profits in the three baseline systems are even less than zero, and are thereby not even comparable with the proposed near-optimal algorithm. We can also observe from Figure 3 that Baseline 1 performs well when the unit energy price is low. This is because the energy cost is much lower than the cost penalty for large average service request response time in this case, and Baseline 1 minimizes the latter cost penalty by running every core at its maximum execution frequency. On the other hand, Baseline 2 performs relatively better when the unit energy price is high since the total energy cost is the dominating factor in this case.

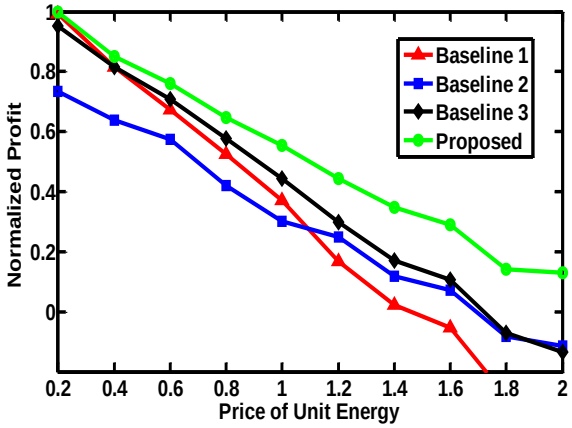

Figure 3: The normalized total profit versus the unit energy price of the proposed near-optimal algorithm and three baseline algorithms.

Figure 4 illustrates the normalized total profit versus the average service request generating rate of the clients on the proposed near-optimal algorithm and three baseline algorithms. In this experiment, we set the unit energy price

Price to be a constant value (equal to 1.0) but change the average service request generating rate of clients. These results show the effectiveness of performing server-level and core-level consolidations. When the average service request generating rate is 0.6, the total profit obtained by the proposed algorithm is 95.6%, 47.6%, and 51.9% higher than Baseline 1, Baseline 2, and Baseline 3, respectively. On the other hand, when the average service request generating rate is 1.0, the total profit obtained by the proposed algorithm is 40.7%, 100.8%, and 19.6% higher than Baseline 1, Baseline 2, and Baseline 3, respectively. When the average service request generating rate becomes higher than 1.2, the profits in some baseline systems become negative.

In fact, the optimal number of turned on cores increases with the increasing in the average service request generating rate. The optimal number of turned on cores is 7 when the average request generating rate is 0.2, and is 20 when the average generating rate is 2.0. The reason is that fewer cores are required for request processing when the average service request generating rate is lower and the static power consumption in the server cluster is reduced in this case.

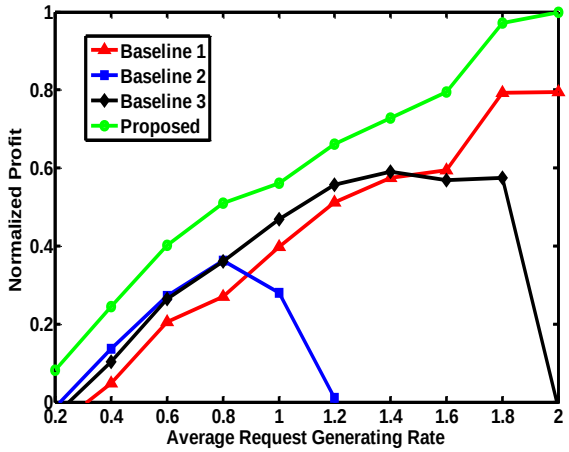

Figure 4: The normalized total profit versus the average service request generating rate of the proposed near-optimal algorithm and three baseline algorithms.

6. Conclusion

In this paper, we consider the problem of SLA-based resource allocation optimization in a server cluster in the cloud computing framework. The objective is to maximize the total profit, which is total price gained from serving the clients, which depends on the average request response time for each client as defined in their utility functions, subtracted by the energy cost of the server cluster. We propose a joint optimization framework comprised of requests dispatching, DVFS for individual cores in the server cluster, as well as core-level and server-level consolidations. Each core in the server cluster is modeled using a CTMDP. We propose a near-optimal hierarchical solution consisting of a central manager and distributed local agents. Each local agent employs linear programming-based CTMDP solving method to solve the DVFS problem for the corresponding core. The central manager solves the request dispatching problem and finds the optimal number of turned on cores and servers for request processing, thereby achieving a desirable tradeoff between service request response time and system power consumption. Experimental results demonstrate that the proposed near-optimal resource allocation and consolidation algorithm consistently outperforms baseline algorithms.

7. References

[1] R. Buyya, "Market-oriented cloud computing: vision, hype, and reality of delivering computing as the 5th utility," in *9th IEEE/ACM International Symposium on Cluster Computing and the Grid* (CCGrid), 2009.

[2] 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", *Commun. of the ACM*, 2010.

[3] L. A. Barroso and U. Holzle, "The case for energy-proportional computing," *IEEE Computer*, 2007.

[4] H. Goudarzi and M. Pedram, "Multi-dimensional SLA-based resource allocation for multi-tier cloud computing systems," *Proc. of IEEE International Conference on Cloud Computing* (CLOUD), 2011.

[5] Y. Wang, S. Chen, and M. Pedram, "Service level agreement-based joint application environment assignment and resource allocation in cloud computing systems," to appear in *Proc. of IEEE Green Technologies Conference* (GreenTech), 2013.

[6] R. Buyya and M. Murshed, "GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *Concurrency and Computation Practice & Experience*, 2002.

[7] K. Krauter, R. Buyya, and M. Maheswaran, "A taxonomy and survey of grid resource management systems for distributed computing," *Software Practice and Experience*, 2002.

[8] Z. Liu, M. S. Squillante, and J. L. Wolf, "On maximizing service-level-agreement profits," in *3rd ACM Conference on Electronic Commerce*, 2001.

[9] L. Zhang and D. Ardagna, "SLA based profit optimization in autonomic computing systems," in *2nd Int. Conf. on Service Oriented Computing*, 2004.

[10] D. Ardagna, M. Trubian, and L. Zhang, "SLA based resource allocation policies in autonomic environments," *Journal of Parallel and Distributed Computing*, 2007.

[11] A. Chandra, W. Gongt, and P. Shenoy, "Dynamic resource allocation for shared clusters using online measurements," *International Conference on Measurement and Modeling of Computer Systems* (SIGMETRICS), 2003.

[12] T. Burd, T. Pering, A. Stratakos, and R. Brodersen, "A dynamic voltage-scaled microprocessor system," *IEEE International Solid-State Circuits Conference Digest of Technical Papers*, 2000.

[13] http://www.intel.com/technology/architecture-silicon/next-gen/

[14] http://support.amd.com/us/Processor_TechDocs/40036.pdf

[15] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing," in *Workshop on Power Aware Computing and Systems* (HotPower'08), 2008.
[16] I. Hwang, T. Kam, and M. Pedram, "A study of the effectiveness of CPU consolidation in a virtualized multi-core server system," in *Proc. of International Symposium on Low Power Electronics and Design* (ISLPED), 2012.
[17] H. Jung and M. Pedram, "Stochastic dynamic thermal management: a Markovian decision-based approach," in *International Conference on Computer Design* (ICCD), 2006.
[18] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, Wiley Publisher, New York, 1994.
[19] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, McGraw-Hill, 3rd edition, 1991.
[20] Z. Zhang, D. Towsley, and J. Kurose, "Statistical analysis of generalized processor sharing scheduling discipline," *ACM SIGCOMM'94 Conf. on Communications Architectures, Protocols and Applications*.
[21] R. E. Bellman, *Dynamic Programming*, Princeton University Press, Princeton, 1957.
[22] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2004.
[23] L. Kleinrock, *Queueing Systems, Volume I: Theory*, New York: Wiley, 1975.
[24] E. D. Andersen and K. D. Andersen, "The MOSEK interior point optimizer for linear programming: an implementation of the homogeneous algorithm," in *High Performance Optimization*, pp. 197 – 232. Kluwer Academic, 2000.
[25] D. Z. Chen, O. Daescu, Y. Dai, N. Katoh, X. Wu, and J. Xu, "Efficient algorithms and implementations for optimizing the sum of linear fractional functions, with applications," *Journal of Combinatorial Optimization*, 2005.