On-Line Real-Time Service Allocation and Scheduling for Distributed Data Centers

Shuo Liu Gang Quan
Electrical and Computer Engineering Department
Florida International University
Miami, FL, 33174
{sliu005, gang.quan}@fiu.edu

Shangping Ren
Computer Science Department
Illinois Institute of Technology
Chicago, IL, 60616
ren@iit.edu

***Abstract*—With the prosperity of Cluster Computing, Cloud Computing, Grid Computing, and other distributed high performance computing systems, Internet service requests become more and more diverse. The large variety of services plus different Quality of Service (QoS) considerations make it challenging to design effective allocate and scheduling algorithms to satisfy the overall service requirements, especially for distributed systems. In addition, energy consumption issue attracts more and more concerns. In this paper, we study a new energy efficient, profit and penalty aware allocation and scheduling approach for distributed data centers in a multi-electricity-market environment. Our approach efficiently manages computing resources to minimize the processing and transferring energy dollar cost in an electricity price varying environment. Our extensive experimental results show the new approach can significantly cut down the energy consumption dollar cost and achieve higher system's retained profit.**

***Keywords*-real-time scheduling; service-oriented; distributed data center; TUF; retained profit;**

I. Introduction

Internet services become more pluralism with the continuous developments of distributed systems. The large variety of services' subcomponents, as well as different combinations of computing resources distributed widely on the Internet, make requests hard to be satisfied, especially in a real-time manner. The dramatically increased service requests and their complexity demand high computation time and energy consumption. Besides achieving the desired Quality of Services, today's service providers also need to take care of their processing efficiency and energy consumption issue. The drastically increased service requests these years directly results in the electricity power consumption sky-rocketing. The energy consumption has become a severe issue across the entire information and communications technology sector [1]. The electricity consumption consumed by data centers was accounted for approximately 1.2% of total consumed by United States in 2005 [2], [3], [4]. 0.4% of total electricity consumption in broadband-enabled countries was consumed by transmission and switching networks [1], [5]. The increased demand on data center does increase the service providers' revenue, but on the other hand, service providers must also make sure to have their energy consumption expense in control.

In North America market, the electricity prices vary a lot based on the different power generating technology, delivery method and coverage. Even though in the same region, the prices may vary significantly during a day. In order to save the construction and electricity investment, many service provider companies, which provides storage, processing, or other services, build their data centers in different locations where there is less population and near to the power plant. Take Google as an example, it has data centers in Portland, Houston, and Atlanta, etc. This helps to ensure the power supply and at the same time bring down their energy consumption dollar cost.

Several researches have been conducted for guaranteeing QoS with energy consumption minimization on data centers. In [6], the authors modeled the service system based on the Queueing Theory. By optimizing the task allocation and the number of powered on servers in each data center, their method successfully reduces the electricity cost in a multi-electricity-market. The authors in [7] presented a Dynamic Voltage Scaling (DVS) based power control mechanism for heterogeneous distributed systems. Their approach properly sets the number of servers that should be powered on and the frequency level at which the working servers should run. Both works model the delay constraint described based on the classic queueing model. While delay constraint can be used to model QoS requirement, it is not as effective for services requests that are sensitive to "timeliness". To this end, the Time Utility Function (TUF) can be used to more accurately describe the timing related QoS requirements.

Time Utility Function is suitable for on-line real-time services, which are mostly soft real-time in nature. TUF, first proposed by Jensen [8], is a function that defines the different values when completing a task at different time (e.g. [9], [10], [11], [12], [13], [14], [15], [16]). The limitation of employing one single TUF, however, is that it can only model the benefit or gain to successfully complete a task before its deadline. It is difficult to model the penalty when a time sensitive task misses its deadline, or sometimes, when such a task has to be aborted. To this end, Yu et

978-0-7695-4462-5/11 $26.00 © 2011 IEEE
DOI 10.1109/SCC.2011.105

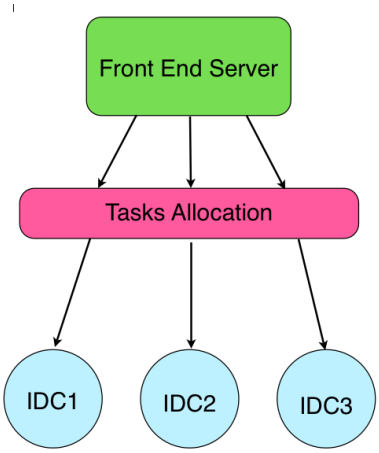


Figure 1. A Simple System Architecture

al. propose to use two TUFs, one to define the gain a system may get when it successfully completes processing a task and another one to reflect the penalty the server may suffer from when a task misses its deadline or is aborted. Several work based on this model have been conducted and the results show the good management of the real-time tasks [18].

In this paper, we propose a new on-line real-time service allocation and scheduling algorithm for distributed data centers in a multi-electricity-market environment. We use a model similar to that in [17] to model the processing cost at data centers and data transferring cost in Internet service provider's networks. Based on this model, we developed a scheduling and allocation method to balance the trade off between the system's retained profit and system performance. The experimental results show that our method can effectively reduce the power consumption and, at the same time, increase the system's retained profit. It is worth mentioning that our work is suitable for both homogeneous and heterogeneous data centers and can be easily extended to accommodate more complex service requests.

The remainder of the paper is organized as follows. Section II introduces the preliminaries. Section III gives our proposed global-local task allocation and scheduling. Experimental results are shown in section IV. Conclusion will be given at the end in section V.

II. System Model

Our system architecture consists of a front end server, a task allocator, and several data centers at different locations. Figure. 1 shows the overall system architecture. The front end server receives the service requests from the Internet. The task allocator assigns the tasks according to some metrics to different data center locations. The services are processed and hosted in the data centers. The energy consumption includes the energy consumption for the computation in data centers and that for data transferring during the allocation process.

The real-time tasks, similar to the one in [17], consists of a single sequence of randomly arrived real-time tasks $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$, with parameters of τ_i defined as follows.

- S_i: The size of the data that supports the task to be processed;
- $[B_i, W_i]$: The best case execution time B_i and the worst case execution time W_i of τ_i;
- D_i: The relative deadline of τ_i;
- $f_i(T)$: The probability density function for the execution time of τ_i;
- $G_i(t)$: The profit TUF, which represents the profit accrued when a task is completed at time t (relative to its arrival time). We assume $G_i(t)$ is a non-increasing unimodal function before its deadline, i.e. $G(t_i) \geq G(t_j)$ if $t_i \leq t_j$, and $G_i(t) = 0$ if $t > D_i$.
- $L_i(t)$: The penalty TUF, which represents the penalty suffered when a task is discarded at time t (relative to its arrival time). We assume $L_i(t)$ is a non-decreasing unimodal function before its deadline, i.e. $L(t_i) \leq L(t_j)$ if $t_i \leq t_j$, and a task is immediately discarded once it missed its deadline.
- $PC_i(t)$: The processing energy consumption.
- $TC_i(t)$: The data transferring energy consumption.
- $Pr(t)$: The electricity price in a multi-electricity-market, which varies with locations and times.

For the power consumption in processing and data transferring step, we used the model provided in [1], the processing power consumption is given in Equation 1:

$$E_S = 1.5 \times T_S \times P_S. \quad (1)$$

where T_S is the time for the server to finish the task processing, P_S is the power of the server. The factor of 1.5 is used to account for the cooling energy consumption.

The energy consumed by the data transferring in Internet service provider's network is give in Equation. 2

$$E_I = 6\times(\frac{3P_{es}}{C_{es}}+\frac{P_{bg}}{C_{bg}}+\frac{P_g}{C_g}+\frac{2P_{pe}}{C_{pe}}+\frac{2\times 9P_c}{C_c}+\frac{8P_w}{2C_w}). \quad (2)$$

where P_{es} is the Ethernet switches' power, P_{bg} is broadband gateway routers routers' power, P_g is the power of data center gateway routers, P_{pe} is the power of provider edge routers, P_c and P_w are the power of core routers and Wavelength Division Multiplexed (WDM) fiber links transport equipment, respectively. C_{es}, C_{bg}, C_g, C_{pe}, C_c, and C_w are the capacities of the corresponding equipment in bits per second. The details of this power model can be found in [1]. They per-bit energy consumption of transmission and switching for a public distributed system was estimated to be around $2.7\mu J/b$.

The tasks in the system are associated with a profit function, a penalty function, a processing energy cost function, and an Internet data transferring energy cost function

with function values varying with time. Therefore, whenever a task is allocated to the data centers, it incurs a data transferring cost. While the tasks are being executed they have a potential to gain profit, it also has a potential to encounter a penalty at a later time. Once a task starts its execution, a processing energy cost is incurred. The system performance is therefore evaluated by its retained profit, which is the total profit subtracts the penalty and energy costs.

Our problem can be formally defined as: *Given a task set* $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$ *as described above, develop an online real-time allocation and scheduling method in order to maximize the retained profit for distributed data centers.*

Since the task mode has variable execution times, execution time for a specific task is not known deterministically. We do not know if executing the task will lead to positive profit or loss. In order to make a proper task allocation and schedule policy, we need to generate a metric which is called the *expected retained profit* to help us make decisions.

Given a task τ_i with arrival time of r_i, let its predicted starting time be T, expected execution time is C_i. Then the potential profit ($\overline{G_i}(T)$) to execute τ_i can be represented as

$$\overline{G_i}(T) = \int_{B_i}^{D_i-(T-r_i)} G_i(t+(T-r_i))f_i(t)dt. \quad (3)$$

Similarly, the potential loss ($\overline{L_i}(T)$) to execute τ_i can be represented as

$$\overline{L_i}(T) = L_i(D) \int_{D_i-(T-r_i)}^{W_i} f_i(t)dt. \quad (4)$$

Therefore, the expected accrued utility ($\overline{U_i}(T)$) to execute τ_i can be represented as

$$\overline{U_i}(T) = \overline{G_i}(T) - \overline{L_i}(T). \quad (5)$$

To account for the power consumed in the process and transfer procedures, we subtract power consumption dollar costs from the expected accrued utility. The expected retained profit is thus given as below

$$\overline{ER_i}(T) = \overline{U_i}(T) - \overline{PC_i}(C_i) - TC_i(r_i). \quad (6)$$

where PC_i is the dollar cost for processing. TC_i is the transferring dollar cost. C_i is task τ_i's execution time, and r_i is task τ_i's arrival time.

A task can be accepted or chosen for execution when $\overline{ER_i}(T) > 0$, which means that the probability of to obtain positive retained profit is no smaller than that to incur a loss. We can further limit the task acceptance by imposing a threshold (δ) to the expected accrued utility, i.e. a task is accepted or can be chosen for execution if

$$\overline{ER_i}(T) \geq \delta. \quad (7)$$

We call δ as the *expected retained profit threshold.*

Furthermore, since the task execution time is not known a prior, the data centers need to decide whether to continue or abort the execution of a task. The longer the task is executed, the closer the data centers are to the completion point of the task. At the same time, however, the longer the task executes, the higher penalty the system has to endure if the task cannot meet its deadline. To determine the appropriate time to abort a task, we employ another metric, i.e. the *critical point*.

Assume task τ_i starts its execution at T. Then the potential profit at $T' > T$ (i.e.$\widetilde{G_i}(T')$) can be represented as

$$\widetilde{G_i}(T') = \int_1^{D_i-(T-r_i)-(T'-T)} G_i(t+(T-r_i))f_i(t)dt. \quad (8)$$

Similarly, the potential loss at $T' > T$ (i.e.$\widetilde{L_i}(T')$) can be represented as

$$\tilde{L_i}(T') = L_i(D) \int_{D_i-(T-r_i)}^{W_i-T'} f_i(t)dt. \quad (9)$$

Therefore, the expected accrued utility at $T' > T$ (i.e. $\widetilde{U_i}(T')$) can be represented as

$$\widetilde{U_i}(T') = \widetilde{G_i}(T') - \widetilde{L_i}(T'). \quad (10)$$

And the expected retained profit is

$$\widetilde{ER_i}(T') = \widetilde{U_i}(T') - \overline{PC_i}(C_i) - TC_i(r_i). \quad (11)$$

We can make $\widetilde{ER_i}(t_0) = \delta$ and solve for t_0. Then when executing task τ_i to time t_0, the expected profit equals its expected loss plus expected power consumption dollar cost. We call t_0 as the *critical point* for executing task τ_i. Due to the non-increasing nature of $\widetilde{G_i}(t)$, the non-negative nature of $\widetilde{P_i}(t)$, and the constant expected process cost $\overline{PC_i}(t)$ and transfer cost $TC_i(t)$, $\widetilde{ER_i}(t)$ is monotonically decreasing as t increases. Therefore, it is not difficult to see that the continuous execution of τ_i beyond the critical point will more likely bring a loss rather than a positive profit.

III. Our Approach

In Section. II, we introduced our system model. Our proposed approach is given in details in this section. We employ a global−local policy for task allocation and scheduling. When a new task arrives, the front end server receives it first. Then, it is sent to the task allocator. The allocator decides which data center is the best one for the new arrival task, and assigns the task accordingly. In each data center, the scheduling will be carried out locally for the tasks assigned to the data centers.

A. Global Task Allocation

Task allocation part works when a new task arrives. It either assigns the task to one of the data centers, or rejects the task which is estimated that they can not meet the QoS requirement. The details of the task allocation algorithm

Algorithm 1 TASK ALLOCATION

Input: Let τ_i be the new arrival task, and let ta_{ij}, $\overline{C_{ij}}$, ts_{ij} be the arrive time, expected execution time and estimated starting time of τ_i in data center DC_j, respectively. Let current time be t. ER_{ij} is the estimated retained profit of the task at data center j. Let the expected utility density threshold be δ.

if A new task, i.e. τ_i arrives **then**
  for Location $j = 1$ to n **do**
    Generate the speculated execution order for τ_i in DC_j, and get its ts_{ij};
    Calculate its ER_{ij} based on its ts_{ij} for each data center;
  end for
  Find the maximum ER_{ij}.
  if $MaxER_{ij} > \delta$ **then**
    Assign the task to the location j;
  end if
  if $MaxER_{ij} \leq \delta$ **then**
    Reject τ_i;
  end if
end if

Algorithm 2 SPECULATED EXECUTION ORDER

Input:Let $\Gamma = \{\tau_1, \tau_2, ..., \tau_k\}$ be the accepted tasks in the ready queue, and let r_i, $\overline{C_i}$ represent the arrival time and expected execution time of τ_i. Let the current time be t

Output: The new list $\Gamma' = \{\tau'_1, \tau'_2, ..., \tau'_k\}$ with the speculated execution order and their corresponding expected retained profit $\widehat{ER}(\tau'_j)$ for $\tau'_j, 1 \leq j \leq k$.

if A task τ_0 is being executed **then**
  $T = r_0 + \overline{C_0}$;
else
  $T = t$;
end if
while Γ is not empty **do**
  for Each task C in Γ **do**
    Calculate $\overline{ER_i}(T)$ based on equation (3), (4), (5), and (6);
  end for
  Select τ_j with the highest $\overline{ER_j}(T)$;
  Add τ_j to the end of Γ';
  $\widehat{ER}(\tau'_j) = \overline{ER_j}(T)$;
  $T = T + \overline{C_j}$;
  Remove τ_j from Γ;
end while

are shown in Algorithm 1. Suppose we have n data center locations.

When a new task arrives, a speculated execution order of the tasks allocated to a specific processor is generated. Based on this order, the new task has an estimated starting time at this data center. Using this time and the task's arrival time, we can calculate out its potential profit and penalty, its processing energy consumption dollar cost, and the data transferring energy consumption dollar cost during the allocation procedure. The combination of these values gives the expected retained profit of the task. The highest expected retained profit is selected and the task is assigned to the corresponding location. If the highest ER is smaller than the threshold, the task is rejected immediately. Otherwise, it has a high possibility to make the system suffer from a loss. The speculated order is generated by using Algorithm. 2.

When generating the speculated execution order we first calculate the expected retained profit for each task in the ready queue based on the expected finishing time of the current running task. Then the task with the largest one is assumed to be the first task that will be executed after the current task is finished. Based on this assumption, we then calculate the expected retained profit for the rest of the tasks in the ready queue and select the next task. This process continues until all tasks in the ready queue are put in order. When completed, we essentially generate a speculated execution order for the tasks in the ready queue.

B. Local Task Scheduling

When tasks arrive at the data centers, the scheduling follows the UA criteria, which try to maximize the accrued utility of the system by successfully completing the tasks in time, to schedule the tasks.

The details of our scheduling is described in Algorithm3. The scheduling algorithm works at every scheduling point, which includes new tasks arrival, tasks completion, tasks critical time, and task's deadline missing point. When there is a new job arrives, the scheduler fist checks its expected retain profit at the time when current running task is expected to be finished. If the expected retained profit is larger than the threshold, it is accepted. if not, it is rejected directly. After adding the new task into the ready queue of the data center, the scheduler generates a speculated execution order by using Algorithm 2, and based on this order removes the tasks that can not satisfy the system's requirement. This step is similar to the one in task allocation procedure. The difference is that in local scheduling, the scheduler has to take care of the other waiting tasks to see if some of them need to be removed or not. When a task finishes its execution, it contributes to the system with a positive profit. The finished task also causes processing and transferring cost. At this time instance, the scheduler selects a new task which has the highest expected retained profit to run. The scheduler works the same when the task reaches to a task's

Algorithm 3 LOCAL TASK SCHEDULING

Input: Let $\{\tau_1, \tau_2, ..., \tau_k\}$ be the accepted tasks in the ready queue, and let $\overline{C_i}$ be the expected execution time of τ_i, and ER_i be the expected retained profit. Let current time be t and let τ_0 be the task currently being executed. Let the expected utility threshold be δ.

if A new task, i.e. τ_n arrives **then**
  Accept τ_n if $\overline{ER_n}(\overline{C_0}) > \delta$;
  Reject τ_n if $\overline{ER_n}(\overline{C_0}) \leq \delta$;
end if
if τ_n is added into the ready queue, generate the speculated order for the data center; **then**
  Remove the tasks whose $ER \leq \delta$;
end if

if τ_0 is completed **then**
  Choose the highest expected retain profit task τ_i to run.
  Remove τ_j in the ready queue if $\overline{ER_j}(\overline{C_i}) \leq \delta$;
end if

if t = the critical time of τ_0, or τ_0 misses its deadline **then**
  Abort τ_0 immediately;
  Choose the highest expected retain profit task τ_i to run.
  Remove τ_j in the ready queue if $\overline{ER_j}(\overline{C_i}) \leq \delta$;
end if

critical time or at deadline missing point. New task selection followed by a scheduling point check. In this step, scheduler further removes the tasks that can not satisfy the system's requirement at the selected task's estimated finishing time.

Our new approach's performance is investigated using simulation. The results are given in next section.

IV. EXPERIMENT

We use simulation experiments to investigate the performance of our proposed approach. Two representative approaches were implemented and compared. One is called the "Optimized" approach, which is our proposed one. The second is the naive approach, called "Averaged". In the naive approach, at the task allocation step, the tasks are allocated according to the total expected execution times summation of different data centers. The new coming task is assigned to the data center which has the shortest expected execution time summation. At each data center, the tasks are executed in a first-come-fist-serve order.

A. Experiment Set Up

We evaluate our proposed method's performance based on several known Google's IDC locations with real-life electricity prices (Houston, Texas, Mountain View, California, and Atlanta, Georgia). The test cases in our experiments were randomly generated. Specifically, S, B, W, and D were randomly generated such that they are uniformly distributed within interval of $[1MegaByte, 2MegaByte]$, $[1, 10]$, $[30, 50]$, and $[40, 50]$, respectively. The execution time of a task is assumed to be evenly distributed between interval of [B,W], i.e. $f(t) = \frac{1}{W-B}$. G, L were assumed to be linear functions, i.e. $G(t) = -a_g(t - D)$ in the range of $[0, D]$ and $L(t) = a_l(t - t_a)$, where t_a is task's arrival time. The gradient for $G(t)$ and $L(t)$, i.e. a_g and a_l were randomly picked from the interval of $[0.5, 1]$ and $[0.1, 0.3]$, respectively. The power P of the homogeneous servers is set to be $1KW$ for the computing ease. The per-bit transferring energy cost is 2.7 $\mu J/bit$. $PC(t_e) = 1.5 \times t_e \times P \times Pr(t_s)$, where t_e is the execution time, t_s is the time when the task starts processing. $TC(t_a) = 6 \times (\frac{3P_{es}}{C_{es}} + \frac{P_{bg}}{C_{bg}} + \frac{P_g}{C_g} + \frac{2P_{pe}}{C_{pe}} + \frac{2\times 9P_c}{C_c} + \frac{8P_w}{2C_w}) \times S \times Pr(t_a)$, which can be simplified as $TC(t_a) = 2.7\mu J/bit \times S \times Pr$. We use arrival time here because once the task arrives, it will be immediately allocated or rejected. Task release times' intervals follow the exponential distribution with $\mu = 2$. Pr we used is shown in Figure. 2 comes from [6], they simulate their approach with the real multi-electricity-market price. It tracks the prices of three Google's data center locations. The retained profit threshold δ is set to 0. We conducted several different groups of experiments to study and compare the performances of different approaches. The results are reported as follows.

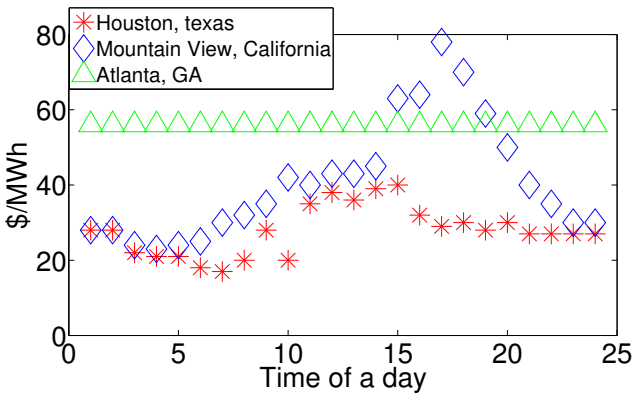


Figure 2. Multi-electricity-market prices for three Google's data center locations.

B. Experiment Results

We first conducted an experiment using a thousand task sets, each with a hundred tasks. We ran our proposed optimized approach and the naive approach on the same task sets. Our new optimized approach aims at increasing the system's accrued retained profit by successfully completing tasks and reducing the tasks' power consumption dollar costs at the same time.

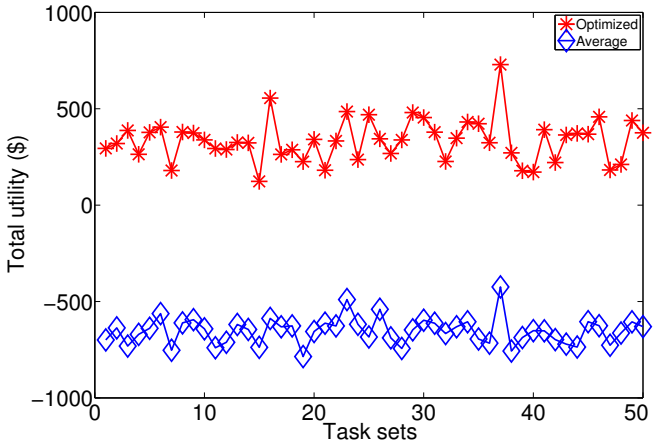


Figure 3. Comparison of accrued retained profit.

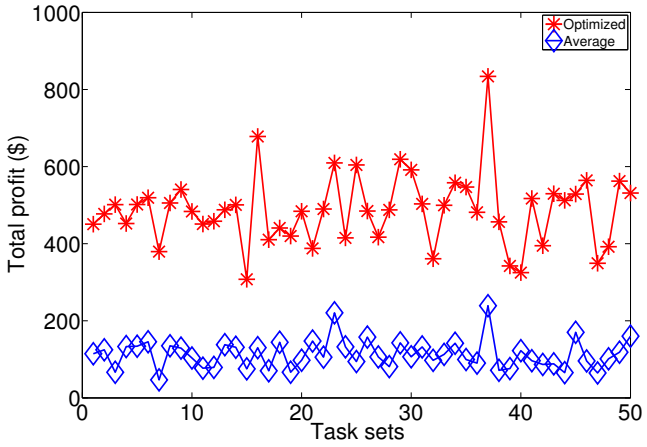


Figure 4. Comparison of profit

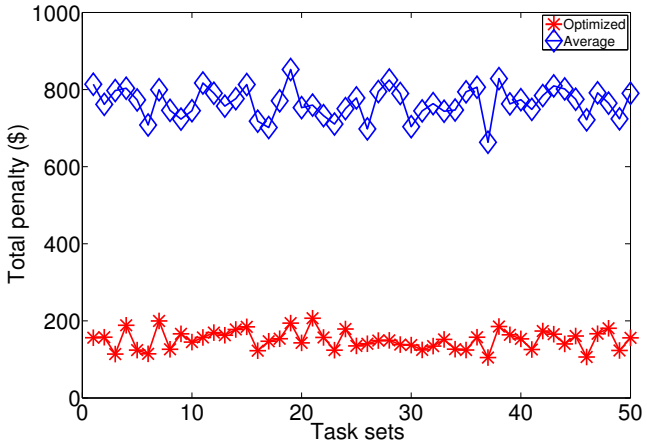


Figure 5. Comparison of penalty

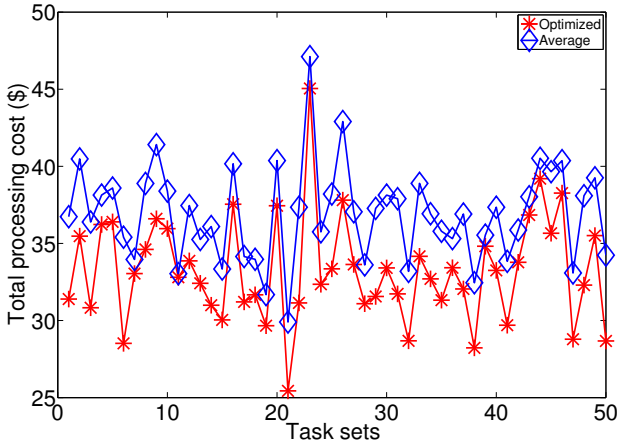
To show the details clearer, Figure 3 displays only the results of the first fifty sets. We can tell that when compared with the naive approach, our optimized approach has a much better performance at attaining higher accrued retained profit.

Figure 4 compares the profits gotten by two approaches. Our optimized approach achieves higher profit since the optimized scheduling finishes more tasks in time successfully than naive scheduling does. Figure 5 indicates the penalty comparison. From Figure 5, we can see that our approach also outperforms the naive approach in the penalty control by discarding less running tasks and missing less deadlines. In our approach, the scheduler carefully accepts potential tasks and judiciously discards running task if it has high possibility to cause loss to the system. Moreover, in the task allocation process, there is an early screening step at the allocation step. The tasks which can not find a proper data center location will be rejected (even the largest expected retained profit for the task is smaller the the threshold δ). The two approaches transfer different number of tasks. This is the reason that why data centers in our optimized approach host less tasks and results in less processing and transferring cost, which is proven by Figure. 6(a) and Figure. 6(b). Figure 6(a) reflects the higher processing efficiency (less hosting tasks and higher profit) of optimized approach. Because of the scheduler's predictability, the tasks that have potential to bring loss are discarded during execution or removed while waiting in the queue. On the contrary, the naive approach always try to finish every executing tasks or keep waiting tasks until they meet their deadline. The lack of forecast induces low processing efficiency. In Figure. 7 we can see our optimized approach completes more tasks than the naive approach does.

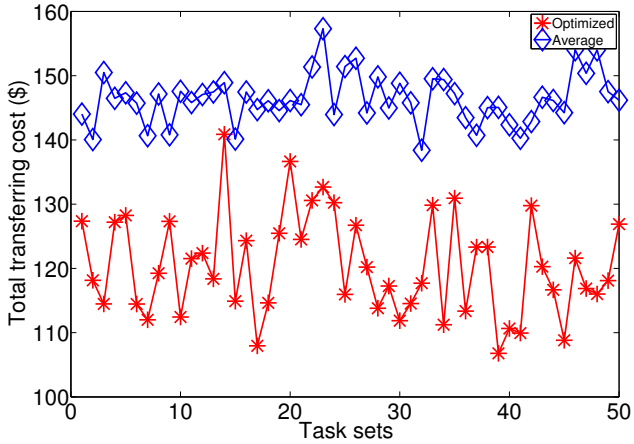
In Figure. 8, the histogram shows the number of assigned tasks of three different data centers. DC3 has the highest average electricity price. The figure shows the average number of tasks assigned to DC3 in optimized approach is significantly smaller than that in naive approach, and also smaller than the number of tasks assigned into DC1 or DC2 in the optimized approach. As shown in the figure, the total number of tasks allocated in optimized approach is less than the number assigned by naive method. As we have explained previously that in the task allocation step, some tasks, which can not find a proper host data center, are rejected instead of being sent to the destinations. Some certain amount of energy is saved here because less task allocations take place in optimized approach. This has already been shown in Figure. 6(b).

V. Conclusion

In this paper, we proposed a novel approach to allocate and schedule tasks for distributed system with the goal to improve the system's profit and optimize the system cost,



(a) Processing power consumption



(b) Transferring power consumption

Figure 6. Power consumption comparison

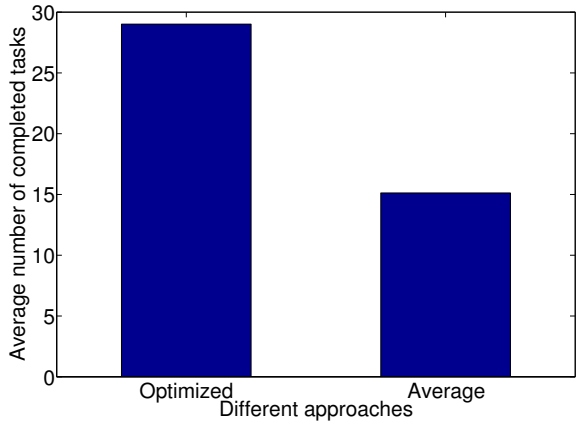


Figure 7. Comparison of number of completed tasks

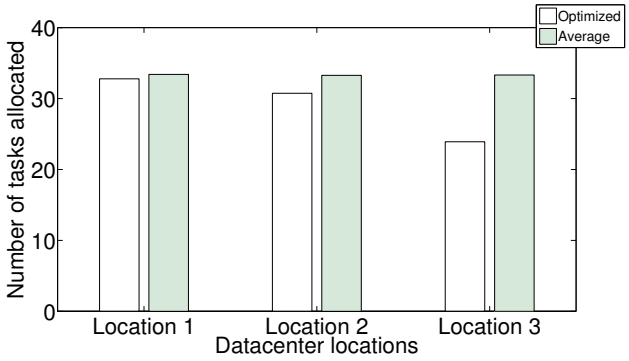


Figure 8. Number of task allocation in different data centers.

including the electricity cost in a multi-electricity-market environment. Different from traditional UA scheduling algorithms, we adopt two Time Utility Functions to model the cost and benefit associating with processing the service request, one for the task's potential profit and the other for its potential penalty. In addition, besides the processing power consumption, we add one more consumption caused by data transferring in internet service provider's network. With the combination of these factors, the approach can carefully accept and allocate tasks, judiciously discard executing tasks and remove pending tasks. Hence our approach can effectively reduce the waste of energy dollar cost and increase the system's retained profit. Moreover, our new proposed approach is suitable for both homogeneous and heterogeneous systems. The formulas in this work are easy to be extended to support more constraints that a distributed system may encounter.

VI. Acknowledgement

This work is supported in part by NSF under projects CNS-0969013, CNS-0917021, CNS-1018108, CNS-1018731, and CNS-0746643.

References

[1] J. Baliga, R. Ayre, K. Hinton, and R. Tucker, "Green cloud computing: Balancing energy in processing, storage, and transport," *Proceedings of the IEEE*, vol. 99, no. 1, pp. 149 –167, jan. 2011.

[2] J. Koomey, "Estimating total power consumption by servers in the u.s. and the world," 2007, Oakland, CA, Analytics Press.

[3] M. Gupta and S. Singh, "Greening of the internet," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, ser. SIGCOMM '03, 2003, pp. 19–26.

[4] GeSI, "Smart 2020: Enabling the low carbon economy in the information age," *The Climate Group London*, 2008.

[5] J. Baliga, R. Ayre, K. Hinton, W. V. Sorin, and R. S. Tucker, "Energy consumption in optical ip networks," *Journal of Lightwave Technology*, vol. 27, pp. 2391–2403, july 2009.

[6] L. Rao, X. Liu, M. Ilic, and J. Liu, "Mec-idc: joint load balancing and power control for distributed internet data centers," in *Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems*, ser. ICCPS '10. New York, NY, USA: ACM, 2010, pp. 188–197. [Online]. Available: http://doi.acm.org/10.1145/1795194.1795220

[7] P. Wang, Y. Qi, X. Liu, Y. Chen, and X. Zhong, "Power management in heterogeneous multi-tier web clusters," in *Parallel Processing (ICPP), 2010 39th International Conference on*, sept. 2010, pp. 385 –394.

[8] E. D. Jensen, C. D. Locke, and H. Tokuda, "A time-driven scheduling model for real-time systems," in *IEEE Real-Time Systems Symposium*, 1985.

[9] R. K. Clark, "Scheduling dependent real-time activities," Ph.D. dissertation, Carnegie Mellon University, 1990.

[10] C. D. Locke, "Best-effort decision making for real-time scheduling," Ph.D. dissertation, Carnegie Mellon University, 1986.

[11] P. Li, "Utility accrual real-time scheduling: Models and algorithms," Ph.D. dissertation, Virginia Polytechnic Institute and State University, 2004.

[12] P. Li, H. Wu, B. Ravindran, and E. Jensen, "A utility accrual scheduling algorithm for real-time activities with mutual exclusion resource constraints," *Computers, IEEE Transactions on*, vol. 55, no. 4, pp. 454–469, April 2006.

[13] H. Wu, "Energy-efficient utility accrual real-time scheduling," Ph.D. dissertation, Virginia Polytechnic Institute and State University, 2005.

[14] H. Wu, U. Balli, B. Ravindran, and E. Jensen, "Utility accrual real-time scheduling under variable cost functions," Aug. 2005, pp. 213–219.

[15] H. Wu, B. Ravindran, and E. D. Jensen, "Energy-efficient, utility accrual real-time scheduling under the unimodal arbitrary arrival model," in *ACM Design, Automation, and Test in Europe (DATE)*, 2005.

[16] D. E. Irwin, L. E. Grit, and J. S. Chase, "Balancing risk and reward in a market-based task service," in *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing*, 2004, pp. 160–169.

[17] Y. Yu, S. Ren, N. Chen, and X. Wang, "Profit and penalty aware (pp-aware) scheduling for tasks with variable task execution time," in *Proceedings of the 2010 ACM Symposium on Applied Computing*, ser. SAC '10. New York, NY, USA: ACM, 2010, pp. 334–339. [Online]. Available: http://doi.acm.org/10.1145/1774088.1774159

[18] S. Liu, G. Quan, and S. Ren, "On-line scheduling of real-time services for cloud computing," *Services, IEEE Congress on*, vol. 0, pp. 459–464, 2010.