Journal of Computer Applications ISSN: 0974 – 1925,
Volume-5, Issue EICA 2012-2, February 10, 2012

Exploiting Resource Overloading Using Utility Accrual Approach for Parallel Data Processing in Cloud

Venkatesa Kumar V
Asst.Professor
Dept. of CSE,
Anna University of Technology
Coimbatore,
E-mail:mail2venkatesa@gmail.com

Siva Malar R
PG Scholar
Dept. of CSE,
Anna University of Technology
Coimbatore,
E-mail:puja2007_malar@yahoo.co.in

Dr.Palaniswami S
Professor & Head,
Dept. of EEE
Govt. College of Technology,
E-mail:joegct81@yahoo.com

Abstract - Parallel Data processing has emerged to be one of the killer applications for Infrastructure-as-a-Service (IaaS) clouds. One of an IaaS cloud's key feature is the provisioning of compute resources on demand. The computer resources available in the cloud are highly dynamic and possibly heterogeneous. Nephele is the first data processing framework to explicitly exploit the dynamic resource allocation offered by today's IaaS clouds for both, task scheduling and execution. Particular tasks of a processing job can be assigned to different types of virtual machines which are automatically instantiated and terminated during the job execution. However, Nephele is not considering resource overload or underutilization during the job execution. In this paper, a novel utility accrual scheduling algorithm is proposed for scheduling the real-time cloud computing services. The most unique characteristics of this approach is that, different from traditional utility accrual approach that works under one single Time Utility Function (TUF), which have two different TUF's, a profit TUF and a penalty TUF – associated with each task at the same time, to model the real-time applications for cloud computing, that need not only to reward the early completions and also to penalize the missing abortions or deadline misses of real-time tasks. To improve the performance of cloud computing, the traditional Utility Accrual (UA) approach is deployed in both Non-Preemptive and Preemptive scheduling. The experimental results shows that the proposed algorithm can outperform the existing Nephele framework and also compare the performance between Non-Preemptive and Preemptive scheduling

Keywords: *Cloud Computing, Task scheduling, Load-balancing, Resource utilization, Performance improvement.*

I. INTRODUCTION

Cloud computing has the potential to dramatically change the landscape of the current IT industry [6], [20], [23]. For companies that only have to process large amounts of data occasionally running their own data center is obviously not an option. Instead, Cloud computing has emerged as a promising approach to rent a large IT infrastructure on a short-term pay-per-usage basis. Operators of so-called IaaS clouds, like Amazon EC2 [2], let their customers allocate, access, and control a set of virtual machines (VMs) which run inside their data centers and only charge them for the period of time the machines are allocated. The VMs are typically offered in different types, each type with its own

characteristics (number of CPU cores, amount of main memory, etc.) and cost. Since the VM abstraction of IaaS clouds fits the architectural paradigm assumed by the data processing frameworks described above, projects like Hadoop [5], a popular open source implementation of Google's MapReduce framework, already have begun to promote using their frameworks in the cloud [29]. Only recently, Amazon has integrated Hadoop as one of its core infrastructure services [3]. However, instead of embracing its dynamic resource allocation, current data processing frameworks rather expect the cloud to imitate the static nature of the cluster environments [16] they were originally designed for, e.g., at the moment the types and number of VMs allocated at the beginning of a compute job cannot be changed in the course of processing, although the tasks the job consists of might have completely different demands on the environment. As a result, rented resources may be inadequate for big parts of the processing job, which may lower the overall processing performance and increase the cost.

One of an IaaS cloud's key feature is the provisioning of compute resources on demand. The computer resources available in the cloud are highly dynamic and possibly heterogeneous. Nephele is the first data processing framework to explicitly exploit the dynamic resource allocation offered by today's IaaS clouds for both, task scheduling and execution. Particular tasks of a processing job can be assigned to different types of virtual machines which are automatically instantiated and terminated during the job execution.

While there exist different interpretations and views on cloud computing [6], [20], [23], it is less disputable that being able to effectively exploit the computing resources in the clouds to provide computing service at different quality levels is essential to the success of cloud computing. For real-time applications and services, the timeliness is a major criterion in judging the quality of service. Due to the nature of real- time applications over the Internet, the timeliness here refers to more than the deadline guarantee as that for hard real-time systems. In this regard, an important performance metric for cloud computing can thus be the sum of certain value or utility that is accrued by processing all real-time service requests.

To improve the performance of Cloud Computing, one approach is to employ the traditional utility accrual (UA) approach [13], [26]. Jensen et al. first proposed to associate each task with a Time Utility Function (TUF), which indicates the task's importance [19]. Specifically, the TUF describes the value or utility accrued by a system at the time when a task is completed. Based on this model, there have been extensive research results published on the topic of UA scheduling [22], [25], [30], [31], [32], [33]. While Jensen's definition of TUF allows the semantics of soft time constraints to be more precisely specified, all these variations of UA-aware scheduling algorithms imply that utility is accrued only when a task is successfully completed, and the aborted tasks neither increase nor decrease the accrued value or utility of the system.

We believe that, to improve the performance of cloud computing, it is important to not only measure the profit when completing a job in time, but also account for the penalty when a job is aborted or discarded. Note that, before a task is aborted or discarded, it consumes system sources including network bandwidth, storage space, and processing power, and thus can directly or indirectly affect the system performance. This is especially true for cloud computing in considering the large possibility of migration

of a task within the clouds for reasons such as the economy considerations [11], [21]. If a job is deemed to miss its deadline with no positive semantic gain, a better choice should be one that can detect it and discard it as soon as possible.

Recently, Yu et al. [34] proposed a task model that considers both the profit and penalty that a system may incur when executing a task. According to this model, a task is associated with two different TUFs, a profit TUF and a penalty TUF. The system takes a profit (determined by its profit TUF) if the task completes by its deadline, and suffers a penalty (determined by its penalty TUF), if it misses its deadline or is dropped before its deadline. It is tempting to use negative values for the penalties, and thus combine both TUFs into one single TUF. However, a task can be completed or aborted and hence can produce either a profit value or a penalty value. Mathematically, if there existed such a single function, it would imply that a single value in its domain was mapped to two values in its range, violating that it is a function. Therefore, one utility function cannot accurately represent both the profit and penalty information when executing a task. There are also some other penalty related models proposed in the literature. For example, Bartal et al. studied the on-line scheduling problem when penalties have to be paid for rejected jobs [7].

This model, however, does not account for the penalty to drop the task before its deadline. However Nephele is not considering resource overload or underutilization during the job execution automatically. In this paper, a novel utility accrual scheduling algorithm is proposed for scheduling the real-time cloud computing services. The most unique characteristics of this approach is that, different from traditional utility accrual approach that works under one single Time Utility Function (TUF), which have two different TUF's, a profit TUF and a penalty TUF – associated with each task at the same time, to model the real-time applications for cloud computing, that need not only to reward the early completions and also to penalize the missing abortions or deadline misses of real-time tasks. To improve the performance of cloud computing, the traditional Utility Accrual (UA) approach is deployed in both Non-Preemptive and Preemptive scheduling.

This paper includes further details on scheduling strategies and extended experimental results. The paper is structured as follows: Section II starts with describing the basic concept of cloud and present the architecture of the Nephele and outline how jobs can be described and executed in the cloud. Section III and IV presents our scheduling approach in details. Experiment results are discussed in Section V and we present the conclusions in Section VI.

II. METRICS AND METHODS

A. Cloud Computing

Cloud computing is Internet-based development and use of computer technology. The cloud is a metaphor for the Internet and is an abstraction for the complex infrastructure it conceals. Cloud computing is a model for enabling ubiquitous, 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. It define in three models Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS)[14]. Figure1 show the architecture of the cloud computing. Cloud computing

system scales applications by maximizing concurrency and using computing resources more efficiently One must optimize locking duration, statelessness, sharing pooled resources such as task threads and network connections bus, cache reference data and partition large databases for scaling services to a large number of users. IT companies with innovative ideas for new application services are no longer required to make large capital outlays in the hardware and software infrastructures. By using clouds as the application hosting platform, IT companies are freed from the trivial task of setting up basic hardware and software infrastructures. Thus they can focus more on innovation and creation of business values for their application services [17]. Some of the traditional and emerging Cloud- based application services include social networking, web hosting, content delivery, and real time instrumented data processing. Each of these application types has different composition, configuration, and deployment requirements. Quantifying the performance of provisioning (scheduling and allocation) policies in a real Cloud computing environment (Amazon EC2 [1], Microsoft Azure [27], Google App Engine [18] for different application models.

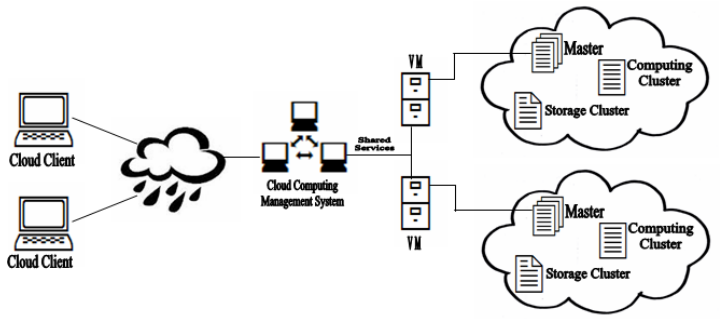

Figure1:Cloud Computing Architecture

Cloud computing also describes applications that are extended to be accessible through the Internet. These cloud applications use large data canter and powerful servers that host Web applications and Web services. Anyone with a suitable Internet connection and a standard browser can access a cloud application.

B. Task Scheduling and Load-balancing Technique

A task is a (sequential) activity that uses a set of inputs to produce a set of outputs. Processes in fixed set are statically assigned to processors, either at compile-time or at start-up (i.e. partitioning). Avoids overhead of load balancing using these load-balancing algorithms. In grid computing algorithms can be broadly categorized as centralized or decentralized, dynamic or static [9], or the hybrid policies in latest trend. A centralized load balancing approach can support larger system. Hadoop system takes the centralized scheduler architecture. In static load balancing, all information is known in advance and tasks are allocated according to the prior knowledge and will not be affected by the state of the system. Dynamic load-balancing mechanism has to allocate tasks to the processors dynamically as they arrive. Redistribution of tasks has to take place when some processors become overloaded [35].

In cloud computing, each application of users will run on a virtual operation system, the cloud systems distributed resources among these virtual operation systems. Every application is completely different and is independent and has no link between each other whatsoever, for example, some require more CPU time to compute complex task, and

some others may need more memory to store data, etc. Resources are sacrificed on activities performed on each individual unit of service. In order to measure direct costs of applications, every individual use of resources (like CPU cost, memory cost, I/O cost, etc.) must be measured. When the direct data of each individual resources cost has been measured, more accurate cost and profit analysis [10].

C. Overview of Nephele Architecture

Nephele is a new data processing framework[28], [8] for cloud environment that takes up many ideas of previous processing frameworks but refines them to better match the dynamic and opaque nature of a cloud.

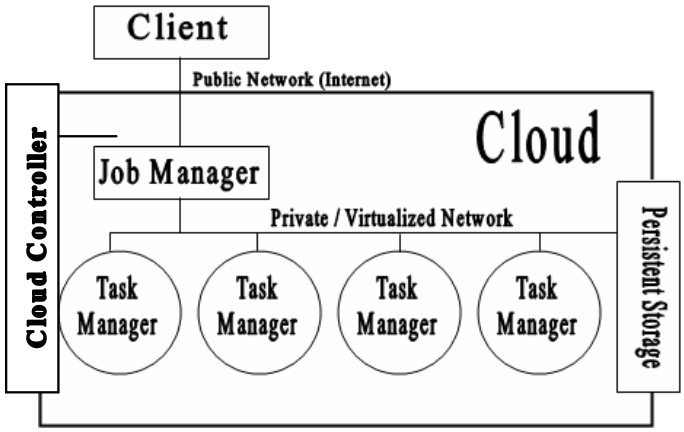

Figure2: Nephele's Architecture

Nephele's architecture follows a classic master-worker pattern as illustrated in Fig. 2.Before submitting a Nephele compute job, a user must start a VM in the cloud which runs the so called Job Manager (JM). The Job Manager receives the Client's Jobs, is responsible for scheduling them and coordinates their execution. It is capable of communicating with the interface the cloud operator provides to control the instantiation of VMs. We call this interface the Cloud Controller. By means of the Cloud Controller, the Job Manager can allocate or deallocate VMs according to the current job execution phase. We will comply with common Cloud computing terminology and refer to these VMs as instances for the remainder of this paper. The term instance type will be used to differentiate between VMs with different hardware characteristics. For example, the instance type "m1.small" could denote VMs with one CPU core, one GB of RAM, and a 128 GB disk while the instance type "c1.xlarge" could refer to machines with 8 CPU cores, 18 GB RAM, and a 512 GB disk.

The actual execution of tasks which a Nephele job consists of is carried out by a set of instances. Each instance runs a so-called Task Manager (TM). A Task Manager receives one or more tasks from the Job Manager at a time, executes them, and after that informs the Job Manager about their completion or possible errors. Unless a job is submitted to the Job Manager, we expect the set of instances (and hence the set of Task

Managers) to be empty. Upon job reception the Job Manager then decides, depending on the job's particular tasks, how many and what type of instances the job should be executed on, and when the respective instances must be allocated/deallocated to ensure a continuous but cost-efficient processing. Our current strategies for these decisions are highlighted at the end of this section. The newly allocated instances boot up with a previously compiled VM image. The image is configured to automatically start a Task Manager and register it with the Job Manager. Once all the necessary Task Managers have successfully contacted the Job Manager, it triggers the execution of the scheduled job.

Initially, the VM images used to boot up the Task Managers are blank and do not contain any of the data the Nephele job is supposed to operate on. As a result, we expect the cloud to offer persistent storage (like eg., Amazon S3 [4]). This persistent storage is supposed to store the job's input data and eventually receive its output data. It must be accessible for both the Job Manager as well as for the set of Task Managers even if they are connected by a private or virtual network.

III. ON-LINE NON-PREEMPTIVE UTILITY ACCRUAL SCHEDULING

In this section we present our on-line non-preemptive scheduling method is used to maximize the accrued gain. Since the execution of a task may gain positive profit or suffer penalty and thus degrade the overall computing performance, judicious Decisions must be made with regard to executing a task, dropping or aborting a task, and when to drop or abort a task. The rationale of our approach is very intuitive, i.e. a task can be accepted and executed only when it is statistically promising to bring positive gain, and discarded or aborted otherwise. Before we introduce the details of our scheduling approach, we first introduce two useful concepts, the *expected accrued utility* and the *critical point.*

A. The expected accrued utility and the critical point

Since the task execution time is not known deterministically, we do not know if executing the task will lead to positive gain or loss. To solve this problem, we can employ a metric, i.e. the *expected accrued utility,* to help us make the decision.
Given a task τ_i with arrival time of *ri,* let its predicted starting time be T. Then the potential profit ($\overline{Gi}$ (T)) to execute τ_i can be represented as

$$\overline{Gi}(T) = \int_{Ri}^{Di-(T-ri)} Gi\left(t + (T - ri)\right) fi(t)dt \qquad (1)$$

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

$$\overline{Li}(T) = \overline{Li}(D) \int_{Di-(T-ri)}^{Wi} fi(t)dt \qquad (2)$$

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

$$\overline{Ui}(T) = \overline{Gi}(T) - \overline{Li}(T) \quad (3)$$

A task can be accepted or chosen for execution when $\overline{Ui}(T) > 0$, which means that the probability of to obtain positive gain 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

$$\overline{Ui}(T) \geq \delta \quad (4)$$

We call δ as the *expected utility threshold.*

Furthermore, since the task execution time is not known a prior, we need to decide whether to continue or abort the execution of a task. The longer we executed the task, the closer we 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.*

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

$$\widetilde{Gi}(T') = \int_{T'-T}^{Di-(T-ri)} Gi\left(t + (T - ri)\right) fi(t)dt \quad (5)$$

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

$$\widetilde{Li}(T') = \text{Li (D)} \int_{Di-(T-ri)}^{Wi-Tr} fi(t)dt \quad (6)$$

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

$$\widetilde{Ui}(T') = \widetilde{Gi}(T') - \widetilde{Li}(T') \quad (7)$$

We can make $\widetilde{Ui}(t_0) = 0$ and solve for t_0. Then when executing task τ_i to time t_0, the expected profit equals its expected loss. We call t_0 as the *critical point* for executing task τ_i. Due to the non-increasing nature of *Gi,* $\widetilde{Ui}(t)$ 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 gain.

Algorithm 1 On-Line Non-Preemptive Accrued Utility Scheduling

---

1. **Input:** Let { $\tau 1$, $\tau 2$, ..., τk} be the accepted tasks in the ready queue, let ri , i = 1, ..., k represent their specific arrival times. Let current time be t and $\tau 0$ be the task currently

being executed. Let the expected utility threshold be δ.
2 **if** t is a scheduling point **then**
3 if t = the critical time of τ0 then
4 Abort the execution of τ0;
5 **end if**
6 **if** A new task, i.e. τj arrives **then**
7 Put τj at the head of ready queue;
8 **end if**
9 Sort tasks in the ready queue based on the recalculated expected accrued utility;
10 Choose the task, i.e. τ1, from the ready queue and start its execution;
11 Remove the tasks with expected accrued utility smaller than δ;
12 **end if**

B. The Scheduling algorithm

Our scheduling algorithm works at scheduling points that include: the arrival of a new task, the completion of the current task, and the critical point of the current task. The detailed algorithm is described in Algorithm 1.

In Algorithm 1, when the time reaches the critical point of the current task, the current active task is immediately discarded and the task with the highest expected accrued utility is selected to be executed. Upon the finish of the current task, the task with the highest expected accrued utility is selected for execution. After the selection of the new task in both of the two cases, the expected accrued utility for the rest of the tasks are re-calculated. The tasks with the expected accrued utility smaller than the threshold value are discarded.

Algorithm 2: SORT the Ready Queue Based on the Recalculated Expected Gain

---

1. **Input:** Let Γr={ τ1 , τ2 , ..., τk} be the accepted tasks in the ready queue, let ri , i = 1, ..., k represent their specific arrival times. Let current time be t and τ0 be the task currently being executed.
2. **Output**: The list of tasks in the ready queue Γ'r={ τ'1 , τ'2 , ..., τ'k} sorted based on their expected gain.
3. Tstart = expected finishing time of τ0 – t;
4. **for** i=0 to k do
5. τ'i = τj where τj ∈ Γr is the task with the largest expected gain assuming it starts at Tstart;
6. Remove τj from Γr;
7. Tstart = Tstart + expected execution time of τ'j;
8. Calculate the following tasks expected utility at time Tstart;
9. **end for**

When a new job comes, it is first inserted at the head of the ready queue, assuming its expected starting time would be the expected finishing time of the current active task.

Based on this starting time, we then can compare its expected utility with the rest of the tasks in the queue. If its expected utility is less than that of the one following it, we re-insert this job to the queue according to its new expected utility. We calculate the new expected utility according to Algorithm 2, by estimating its new expected starting time as the sum of the expected executing time of the leading tasks' in the ready queue. This procedure continues until the entire ready queue becomes a list ordered according to their expected utilities. We remove the ones with expected utility lower than the threshold.

IV. ON-LINE PREEMPTIVE UTILITY ACCRUAL SCHEDULING

In this section, we present a new preemptive scheduling algorithm which belongs to a new family of real-time service oriented scheduling problems. As the complementarily of our previous non-preemptive algorithm [34], real time tasks are scheduled preemptively with the objective of maximizing the total utility this time. Different from the traditional utility accrual scheduling problem that each task is associated with only a single time utility function (TUF), two different TUFs – a profit TUF and a penalty TUF – are associated with each task, to model the real-time services that not only need to reward the early completions but also need to penalize the abortions or deadline issues.
We present a preemptive scheduling heuristics to judiciously accept, schedule, and abort real-time services when necessary to maximize the accrued utility. The new scheduling algorithm has much better performance than an earlier scheduling approach based on a similar model does.

Algorithm 3: On-Line Preemptive Accrued Utility Scheduling

1. Input: Let {τ1, τ2, ..., τk } be the accepted tasks in the ready queue, and let Ci be the expected execution time of τi . Let current time be t and let τ0 be the task currently being executed. Let the expected utility density threshold be δ.
2. if A new task, i.e. τp arrives then
3. Check if τp should preempt the current task or not;
4. if Preemption allowed then
5. τp preempts the current task, and starts being executed;
6. end if
7. if Preemption not allowed then
8. Accept τp if $\frac{\overline{U}p(\overline{C}o)}{\overline{C}o} > \delta$;
9. Reject τp if $\frac{\overline{U}p(\overline{C}o)}{\overline{C}o} \leq \delta$;
10. end if
11. Remove τj in the ready queue if $\frac{\overline{U}j(\overline{C}o)}{\overline{C}j} \leq \delta$;
12. end if
13. if At preemption check point then
14. PREEMPTION CHECKING;
15. end if

16. if τ0 is completed then
17. Choose the highest expected utility density task τi to run.
18. Remove τj in the ready queue if $\frac{\overline{U}j(\overline{C}i)}{\overline{C}j} \leq \delta$;
19. end if
20. if t = the critical time of τ0 then
21. Abort τ0 immediately;
22. Choose the highest expected utility density task τi to run.
23. Remove τj in the ready queue if $\frac{\overline{U}j(\overline{C}i)}{\overline{C}j} \leq \delta$;
24. end if

The details of our scheduling are described in Algorithm 3. There are five main parts in the scheduling. They are the preemption checking, feasibility checking, task selecting, scheduling point checking, and critical point checking. When new tasks are added in to ready queue, not matter whether there is preemption or not, the feasibility checking will work to check if the new ready queue is feasible or not. If any task can not meet the requirement, it will be removed from the ready queue. Scheduling point checking makes sure all the left tasks in the expected accrued utility density task to run when the server is idle. The critical point checking will always monitor the current running task's state to prevent the server wasting time on the non-profitable running task. The preemption checking works when there is a prosperous task wants to preempt the current task. The combination of these parts guarantees to judiciously schedule the tasks for achieving high accumulated total utilities. It is worthy to talk more about the preemption checking part in details, because improper aggressive preemption will worsen the scheduling performance. From Algorithm.4 we can see that if a task can be finished successfully before its deadline even in its worst case, the scheduling will protect the current running task from being preempted by any other tasks. Otherwise, if a prosperous task has an expected accrued utility density which is larger than the current running task's conditional expected accrued utility density by at least a value equals to the pre-set preemption threshold, the preemption is permitted.

Algorithm 4: Preemptive Checking

1. Input: Let τ0 be the task currently being executed, and τp be the task wants to preempt τ0, current time be t, U(τ0,t) be the conditional expected utility density of τ0 at time t, $\overline{C}o'$ is the remaining expected time of τ0. $\overline{U}p(t)$ be the expected utility density of τp;
2. if $\frac{\overline{U}p(t)}{\overline{C}p} > \frac{\tilde{U}(To,t)}{\overline{C}o}$ then
3. Check what is τo's worst case finish time;
4. if τ0 can be finished before its deadline even in the worst case then
5. Preemption not allowed;
6. end if
7. if τo's worst case will miss its deadline then
8. Preemption allowed;

9. end if
10. end if

The feasibility check is one more part deserves detail description. In this part, scheduling simulates the real execution sequence for the left tasks in ready queue and check following this sequence, if all of them can satisfy the requirement or not. The thing needs to be discussed is how to determine the sequence of the left tasks. From equation (1), (2) and (3), we can clearly see that the expected utility of running a task depends heavily on variable T, i.e. the time when the task can start. If we know the execution order and thus the expected starting time for tasks in the ready queue, we will be able to quantify the expected utility density of each task more accurately. In algorithm.5, we show our utility metric based on a speculated execution order of the tasks in the ready queue.

The general idea to generate the speculated execution order is as follows. We first calculate the expected utility density for each task in the ready queue based on the expected finishing time to 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 utilities 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 and, at the same time, calculate the corresponding expected utility density for each task.

Algorithm 5: Preemptive Checking

---

1. **Input:** Let Γ={ τ1 , τ2 , ..., τk} be the accepted tasks in the ready queue, let ri , $\overline{C}i$ represent the arrival time and expected execution time of τi. Let the current time be t.
2. **Output**: The new list Γ'={ τ'1 , τ'2 , ..., τ'k} with the speculate execution order and their corresponding expected utility density $\hat{U}(T'j)$ for τ'j, 1≤ j ≤ k.
3. if A task τ0 is being executed then
4. T = r0 + $\overline{C}0$;
5. else
6. T=t;
7. end if
8. While Γ is not empty do
9. for Each task C in Γ do
10. Calculate $\frac{\overline{U}i(T)}{Ci}$ based on equation (1), (2)and (3);
11. end for
12. Select τj with the highest $\frac{\overline{U}j(T)}{Cj}$;
13. Add τj to the end Γ';
14. $\hat{U}(Tj) = \overline{U}j(T)$;

15. T = T + $\overline{C}j$;
16. Remove τj from Γ;
17. end while

In the next section, we investigate and compare the performance of the algorithm using simulation under a variety of different conditions.

V. RESULTS AND DISCUSSION

In this section, we use experiments to investigate the performance of our proposed algorithm.

A. Experiment set up

The test cases in our experiments were randomly generated. Specifically, B, W, and D were randomly generated such that they are uniformly distributed within interval of [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_lt. The gradient for G(t) and L(t), i.e. a_g and a_l were randomly picked from the interval of [4, 10] and [1, 5], respectively. Task release times' intervals follow the exponential distribution with μ = 2. The utility threshold δ is set to 0. We conducted three different groups of experiments to study and compare the performance of different approaches under different conditions. The results are reported as follows.

B. Experimental results

We first constructed 1000 task sets, each of which consists of 20 tasks. Figure 1, Figure 2, and Figure 3 plot the accrued utility, accrued profit, as well as the accrued penalty for three different approaches: Non Pre-emptive, Preemptive and Nephele. For ease of presentation, we only show 50 sets of results in the figures. The horizontal axis is the index of the experiment sets.

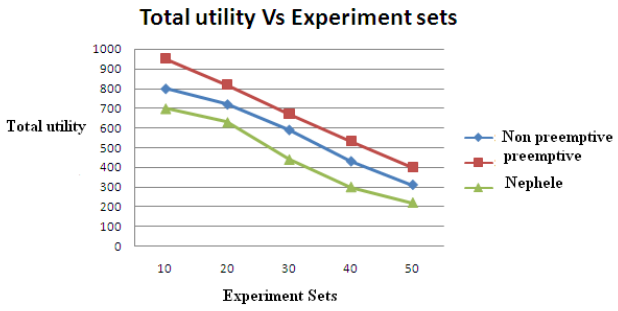

Figure 3: Accrued Utility

The above graph shows that the variation between the total utility and the experiment sets. From that we can know, how the values of total utility will increased to the corresponding values of experiment sets. And this graph shows that, preemptive results are having higher total utility than the Non preemptive and execution graph of the Nephele.

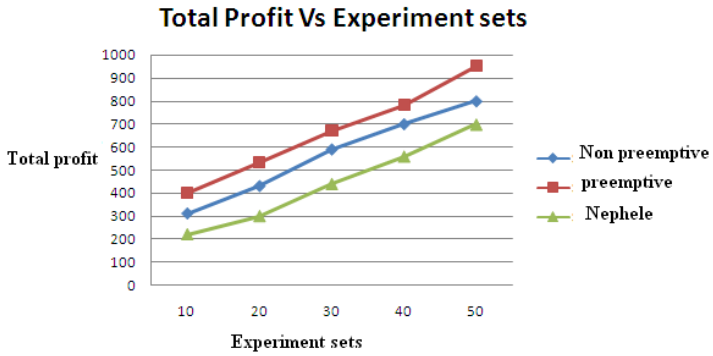

Figure 4: Accrued Profit

The above graph shows that the variation between the total profit and the experiment sets. From that we can know, how the values of total profit will increased to the corresponding values of experiment sets. And this graph shows that, preemptive results are having higher profit than the Non preemptive and execution graph of the Nephele.

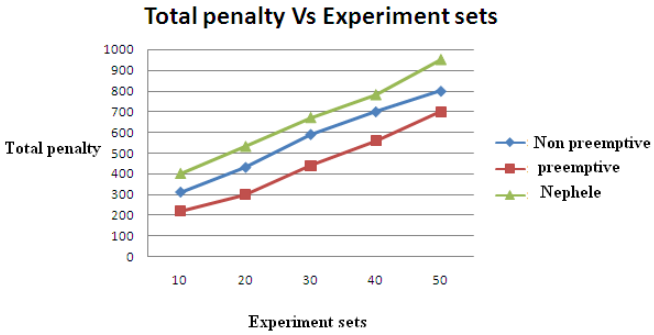

Figure 5: Accrued Penalty

This graph shows that the variation between the total penalty and the experiment sets. From that we can know, how the values of total penalty will increased to the corresponding values of experiment sets. And this graph shows that, Non-preemptive results are having higher profit than the preemptive and execution graph of the Nephele.

VI. CONCLUSION

The popularity of the Internet has grown enormously, which has presented a great opportunity for providing real-time services over the Internet. We have discussed the challenges and opportunities for efficient parallel data processing [12] in cloud environments and presented Nephele, the first data processing framework to exploit the dynamic resource provisioning offered by today's IaaS clouds. We have described

Nephele's basic architecture and presented a performance comparison to the well-established data processing framework Hadoop. The performance evaluation gives a first impression on how the ability to assign specific virtual machine types to specific tasks of a processing job, as well as the possibility to automatically allocate/deallocate virtual machines in the course of a job execution, can help to improve the overall resource utilization and, consequently, reduce the processing cost. The on-line real-time service system should be compatible with preemption in respect that it is necessary and befitting for nowadays' service requests. Our experimental results clearly show that our proposed preemptive scheduling algorithm is effective in this regard.

In this paper, we present a novel utility accrued scheduling approach which accounts for not only the gain by completing a real-time task in time but also the cost when discarding or aborting the task. Our scheduling algorithm carefully chooses highly profitable tasks to execute, aggressively removes tasks that potentially lead to large penalty, and judiciously allows preemptions. This paper can be viewed as the extended [15] version of Nephele. It is also a significant improvement compared to non-preemptive scheduling [24] in which, the preemptive approaches better than the non-preemptive counterpart. Our extensive experimental results clearly show that our proposed preemptive algorithm can outperform the non-preemptive approach.

REFERENCES

1. Amazon Web Services LLC, "Amazon Elastic Compute Cloud (Amazon EC2)," http://aws.amazon.com/ec2/,2011.
2. Amazon Web Services LLC, " Amazon Elastic MapReduce," http:// aws. amazon .com/ elasticmapreduce/, 2011.
3. Amazon Web Services LLC, "Amazon Simple Storage Service," http : / / aws .amazon.com/s3/, 2011.
4. The Apache Software Foundation "Welcome to Hadoop!" http :// hadoop . apache.org/, 2011.
5. M. Armbrust, A. Fox, R . Griffith, A. D. Joseph, R. Katz,A. Konwinski,G. Lee, D. Patterson, A. Rabkin, I. Stoica,and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," UC Berkeley, 2009.
6. Y.Bartal, S. Leonardi, A. Marchetti - Spaccamela, J. S. gall, and L. Stougie, "Multiprocessor scheduling with rejection," in Proceedings of SODA, 1996, pp. 95 - 103.464
7. D. Battre´ , S. Ewen, F. Hueske, O. Kao, V. Markl, and D. Warneke, "Nephele / PACTs: A Programming Model and Execution Frame- work for Web-Scale Analytical Processing," Proc. ACM Symp. Cloud Computing (SoCC '10), pp. 119-130, 2010.
8. H.J. Braun, T. D.and Siegel, N. Beck, L.L. Blni, M. Maheswaran, A.I.Reuther, J.P. Robertson, M.D. Theys, and B. Yao, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," Journal of Parallel and Distributed Computing, vol. 61, no. 6, 2001, pp.810–837.
9. J.A Brimson, Activity Accounting: An Activity-based Costing Approach, John Wiley & Sons, 1991.
10. F. Casati and M. Shan, "Definition, execution, analysis and optimization of

composite e-service," IEEE Data Engineering, 2001.
11. R. Chaiken , B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S.Weaver, and J. Zhou, "SCOPE: Easy and Efficient Parallel Processing of Massive Data Sets," Proc. Very Large Database Endowment, vol. 1, no. 2, pp. 1265-1276, 2008.
12. R. K. Clark, "Scheduling dependent real-time activities," Ph.D. dissertation, Carnegie Mellon University, 1990.
13. Cloud computing and distributed computing. http://www.cncloudcomputing.com/.
14. Daniel Warneke and Odej Kao "Exploiting Dynamic Resource Allocation For Efficient Parallel Data Processing in the Cloud" IEEE Transactions on Parallel and Distributed System, Vol.22, No.6, 2011.
15. T. Dornemann, E. Juhnke, and B. Freisleben, "On-Demand Resource Provisioning for BPEL Workflows Using Amazon's Elastic Compute Cloud," Proc. Ninth IEEE/ACM Int'l Symp. Cluster Computing and the Grid (CCGRID '09), pp. 140-147, 2009.
16. D. Goldberg. Genetic Algorithms in Search Optimization and Machine Learning. Reading MA Addison Wesley, 1989.
17. GoogleAppEngine, http://code.google.com/appengine, (accessed 14.01.2011)
18. E. Knorr and G. Gruman, "What cloud computing really means," http://www.infoworld.com, 2010.
19. H. Kuno, "Surveying the e-services technical landscape," in 2nd International Workshop on Advanced Issues of Ecommerce and Web-based Information Systems, 2000.
20. 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.
21. P. Li, H. Wu, B. Ravindran, and E. Jensen, "A utility accrual scheduling algorithm for real-time activities with mutual exclusion resourceconstraints," Computers, IEEE Transactions on, vol. 55, no. 4, pp. 454–469, April 2006.
22. MicrosoftAzure, http://www.microsoft.com/windowsazure (accessed 15.01.2011)
23. D. Warneke and O. Kao, "Nephele: Efficient Parallel Data Processing in the Cloud," Proc. Second Workshop Many Task Computing on Grids and Supercomputers (MTAGS '09), pp. 1-10,2009.
24. T. White, Hadoop: The Definitive Guide. O'Reilly Media, 2009.
25. H. Wu, "Energy-efficient utility accrual real-time scheduling," Ph.D. dissertation, Virginia Polytechnic Institute and State University, 2005.
26. H. Wu, B. Ravindran, and E. Jensen, "Utility accrual scheduling under joint utility and resource constraints," May 2004, pp. 307–.
27. H. Wu, U. Balli, B. Ravindran, and E. Jensen, "Utility accrual real-time Scheduling under variable cost functions," Aug. 2005, pp. 213–219.
28. H. Wu, B. Ravindran, and E. Jensen, "On the joint utility accrual model," April 2004, pp. 124.
29. Y. Yu, S. Ren, N. Chen, and X. Wang, "Profit and penalty aware(pp-aware) scheduling for tasks with variable taskexecution time," in SAC2010 – Track on Real-Time System (RTS'2010).
30. M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy , S. Shenker and I.Stoica, "Job Scheduling for multi-user mapreduce clusters," EECS Department, University of California, Berkeley, Tech. Rep. USB/EECS Ph.D. dissertation, Virginia Polytechnic Institute and State University, 2004.