# Toward Waste-Free Cloud Computing

*Tung Nguyen and Weisong Shi*
*Department of Computer Science*
*Wayne State University*
{nttung,weisong}@wayne.edu

## Abstract

Many previous efforts focus on improving the utlization of data centers. We argue that even highly utilized systems can still waste the resource because of failures and maintenance. Therefore, in this position paper, we call on a new direction of research on waste-free computing. We start with a workable definition of efficiency, then introduce a new efficiency metric called **usage efficiency**, show how to compute it and propose some possible approaches to improve this type of efficiency. We also calculate the metric through the simulation on real failure traces and diverse workflow applications. While the efficiency computed from the traditional method is 100%, it is less than 44% with our new metric which suggests that the current metrics are inadequate to capture the efficiency of a system.

## 1 Introduction

We have witnessed the fast growing of large-scale data centers because of the promise of cloud computing, running either publicly or privately. In this position paper, we argue that computing resources in those facilities are not being used efficiently. They are either *under-utilized* or *doing meaningless work*. Next we will use four examples to support this argument.

Observation 1: Schroeder and Gibson [22] envision that, by 2013, the utilization of a top500 computer (top500.org) may drop to 0% based on their assumption about the growth in number of cores per socket and the checkpoint overhead. This is because the only job that a future system would be busy with is checkpointing due to the fact that the number of failures in future computers would increase proportionally to the number of cores.

Observation 2: In a typical industrial data center, the average utilization is only 20 to 30 percent [4, 6] (due to the QoS guarantee for example). However, such low utilized or idle servers still draw 60% of the peak power [4, 14, 11]. The average CPU utilization of 5000 Google servers in 6 months is in the 10-50% range which is in the most energy inefficient region [3].

Observation 3: In systems that offer infrastructure as a service (IaaS), the virtualization may lead to low utilization too. Basically, virtualization techniques allow us to "cut" a physical machine into isolated virtual machines. The inefficiency comes not only from the overhead of the splitting process itself but also from the mismatching between virtual and real resources especially in heterogeneous systems. For example, an old machine with only 1.5GHz processor can only offer one EC2-like standard instance(1-1.2GHz) and the rest is wasted.

Observation 4: Together with the rapid penetration of multicore architectures, the utilization of these cores becomes a big issue. It's quite common to see that one core is very busy while the other ones are pretty idle. For example, when executing the serial part of a program, only one core is needed and others are idle [12]. Furthermore, the issues of maintaining consistency, synchronization, communication, scheduling between many cores might introduce considerable overhead too.

Based on these observations, we categorize the wasted resources into two types: *wasted under-utilized resource* and *wasted utilized resource*. The first type, *wasted under-utilized resource*, is the difference between the available (physical) resources and the consumed resources. It is caused by the low utilization as in the "Observations 2, 3 and 4". It is used to answer the question: *how many percentages of the available resources are utilized?* The second type, *wasted utilized resource*, is the difference between the consumed resources and the *useful resources* (spent on users' jobs). It is often caused by the administrative or maintenance tasks as in the "Observation 1 and 4". It is used to answer the question: *Among utilized resources, how many percentages are spent on doing useful work?*

In addition, the need for resources such as energy is *substantially increasing* as the time of utility computing is approaching. Most data, applications and computation are moving into Cloud. This trend will obviously increase the need of more powerful data centers which are energy hungry. Silicon Valley Leadership Group has also forecasted this increasing trend in the data center energy use based on the Environmental Protection Agency (EPA) reported earlier in 2007 [2, 10].

As a result, using resource *efficiently* or *minimizing the waste* is becoming a very important issue. Minimizing the wasted under-utilized resources, i.e. maximizing the utilization of resources, has been a hot topic in both academic and industrials, such as data center designs and cloud vendors, in order to improve their sustainability. In this paper, we argue that improving the resource utilization is useful, however, is not enough. We also need to reduce the resource wasted for other purposes such as failures and maintenance (wasted utilized resources). For example, if during its execution, one critical task fails resulting the start-over of the whole job, the resources have been used before the failure are wasted. We envision that those types of resource waste, which are the root of the lower efficiency of computing, should be eliminated significantly. Therefore we introduce **waste-free computing** which considers the efficiency from different angles.

## 2 Efficiency Computation

Essentially, efficiency is a multi-objective problem. Unlike the traditional systems dealing with single-objective optimization problems such as minimizing the execution time, minimizing power consumption or maximizing throughput, etc, efficient systems try to "do more with less". It is obvious that there is a trade-off between objectives. The trade-off between benefits and cost is encapsulated in one metric called efficiency. In this section, we will go over some widely used formulas to calculate the efficiency before deriving our own formula.

Generally, efficiency is defined as follows

$$Efficiency = \frac{Return}{Cost} \tag{1}$$

The $Return$ can be profit, the number of work done, improvement of performance (execution time), throughput, bandwidth, availability, queries per second etc. The $Cost$ is what we spend to obtain the $Return$ such as money, energy consumption, CPU time, storage space, etc.

Depending on specific $Return$ and $Cost$, there are different types of efficiency. Let's consider some examples:

Example 1: in term of CPU time, the CPU efficiency is defined as the CPU time spent on doing useful work, divided by the total CPU time. Assume T is the processing time for each process before context switch could occur and S is the overhead for each content switch. For round robin scheduling with quantum Q seconds:

$$\begin{aligned}\text{CPU Efficiency} &= \frac{T}{T+S} \text{ if } Q > T \\ &= \frac{T}{T + S \times \frac{T}{Q}} \text{ if } S < Q < T \end{aligned} \tag{2}$$

Example 2: in term of energy, there are many levels of efficiency ranging from chip to data center. At processor level, CPU efficiency is defined as the performance (MIPS) per joule or watt [20]. At storage level, storage efficiency is define as number of queries per second (QPS) per joule or watt [23].

At datacenter level, recent work [8, 9] has defined two widely used metrics (Power Usage Effectiveness (PUE) and Data Center Infrastructure Efficiency(DCiE)) to measure the data center infrastructure efficiency and compare the energy efficiency between data centers.

$$PUE = \frac{\text{Total Facility Power}}{\text{IT Equipment Power}} \tag{3}$$

$$DCiE = \frac{\text{IT Equipment Power}}{\text{Total Facility Power}} \times 100\% \tag{4}$$

Arguing that PUE and DCiE "do not allow for monitoring the effective use of the power supplied – just the differences between power supplied and power consumed," Gartner [7, 18] proposed a new metric called Power to Performance Effectiveness (PPE).

$$PPE = \frac{\text{Useful Work}}{\text{Total Facility Power}} \tag{5}$$

Moreover, there is work that considers the combination of efficiency at different levels. For example, in [3], Barroso and Hölzle factorized the Green Grid's Datacenter Performance Efficiency (DCPE) into 3 components to capture different types of energy efficiency such as facility, server energy conversion, and electronic components (from large scale to small scale). They defined energy efficiency metric as (detail computation can be found in [9, 3, 8])

$$\begin{aligned}\text{Energy Efficiency} &= \frac{Computation}{\text{Total Energy}} \\ &= \left(\frac{1}{PUE}\right) \times \left(\frac{1}{SPUE}\right) \times \\ &\left(\frac{Computation}{\text{Total Energy to Electronic Comp.}}\right)\end{aligned} \tag{6}$$

The inefficiency is caused by the waste. Where does waste come from? In the first example, the waste is in CPU time. It comes from the context switch time and the scheduling algorithm. In the second example, the waste is in energy. It comes from unsuitable, inefficient design, architecture, cooling, power conversion and dispensation, etc.

Apart from the previous types of efficiency, we introduce another type of efficiency called **usage efficiency**. With this type, failure is the cause of inefficiency and resource waste. For example, if users submit a job to the system, and it is assigned to unreliable nodes. If some of the nodes fail while executing their job (which is very likely with unreliable nodes), all resources consumed or cost spent on execution before failure are useless. This not only reduces the performance (increase the total execution time) but also increases the total cost. Even worse, this can also affect the schedule of other jobs. Therefore, we introduce a new efficiency metric as the following.

$$\text{Usage efficiency} = \frac{\text{Required Resource}}{\text{Actual Resource Used}} \qquad (7)$$

The objective is to minimize the number of job re-execution or, more precisely, reduce the probability to execute the job again.

## 3 Usage Efficiency Computation

In this section, we are going to compute the proposed usage efficiency. First we need to identify clearly what are `Required Resource` and `Actual Resource Used`.

In our case, the `Required Resource` is the total execution time of a job without failure and the `Actual Resource Used` is the actual execution time. The execution time is considered as a resource. The more time the job is executed, the more resource it consumes. In the perfect system, i.e. there is no failure, this type of efficiency has value of 1. The general equation (7) is rewritten as

$$\text{Usage Efficiency} = \frac{\text{Total exec time w/o failure}}{\text{Real total exec time}} \qquad (8)$$

In the problem, a `job` is composed of $k$ `tasks`. A job is executed on $N$ processing instances (PI) which are either virtual or physical machines.

Let $P_i, (i = \overline{1, N})$ be the probability that the $i^{th}$ PI fails during the job execution time.

Each task has execution time $t_j$ $(j = \overline{1, k})$

For simplicity, we assume that $N > k$ and each PI executes one thread. Therefore, if the PI fails, so does the task running on it. Also, we assume that the failure is of the type fail stop only.

For a job, if one task of a job fails, do we need to re-execute the whole job from the beginning, part of the job or just that task only? Can we reuse the results of the successful, completed tasks? This essentially depends on the dependencies among tasks. With the extremely scale map-reduce like jobs, if one task fails, the system only needs to re-execute it. Unfortunately, not all applications fall into such type. Many of them follow workflow model and there are tasks that are correlated. In such model, we may need to replicate the execution of a task to mask the failure or store the intermediate data to avoid re-execution of the job from the beginning.

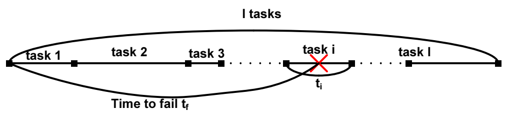

Figure 1: The real execution time of a job.

Assuming that we know the "critical execution path" of the job which the tasks on it decide the total execution time of the job (Figure 1). If any of them is delayed, so does the job. We consider two cases if any of such tasks fail: (1) the execution starts over from the beginning; (2) the execution just restarts the failing tasks. We don't consider replicated execution here. Let $l$ be the length of the path $(1 \leq l \leq k)$. It's also the number of tasks on that path.

### 3.1 Macro-rescheduling: Re-execute the whole job

The probability that all $l$ critical nodes/tasks don't fail during their execution is $\Pi_{i=1}^{l}(1 - P_i)$.

Let Y be a random variable denotes the number of times that the job is re-executed. If *task j is re-executed on the same node after failure*, and the *failure probability $P_i$ of the $i^t h$ PI doesn't change* among executions, Y follows Geometric distribution, and therefore has the expectation as the following.

$$E[Y] = \frac{1}{\Pi_{i=1}^{l}(1 - P_i)}$$

Without failure, the total execution time of the job is the sum of all tasks on the critical path which is $T = \Sigma_{i=1}^{l} t_i$.

With failure, in this case, the job is start-over from the beginning as shown in Figure 1. The probability that at least one critical task fails equals to the probability that the job is re-executed

$$P_{\text{at least 1 fail}} = P_{\text{re-executed}} = 1 - \Pi_{i=1}^{l}(1 - P_i)$$

It is noteworthy that although the time to fail is uniformly distributed within $t_j$ for individual critical task j, it is different across tasks.

$\Rightarrow$ the expected time to fail for each execution before the first success of the job is $\frac{\sum_{j=1}^{l} t_j}{2}$.

Let T be the real execution time of the job, and $t_{f_n}$ be the time to fail at the $n^{th}$ execution of the job.

$$T = P_{\text{at least 1 fail}} \times \sum_{n=1}^{Y} t_{f_n} + \Pi_{i=1}^{l}(1-P_i) \times \sum_{j=1}^{l} t_j$$

$\Rightarrow$ the expected execution time of the job is

$$E[T] = P_{\text{at least 1 fail}} \times \left( E[Y] \times \frac{\sum_{j=1}^{l} t_j}{2} \right) + \Pi_{i=1}^{l}(1-P_i) \times \sum_{j=1}^{l} t_j$$

## 3.2 Micro-rescheduling: Re-execute the failing tasks

In this case, the real execution time of the job is simply the sum of that of its individual tasks on the critical path. Let $X_i$ be a random variable denotes the number of times that task $i^{th}$ is re-executed.

$$T = \sum_{i,j=1}^{l} T_{ij} = \sum_{i,j=1}^{l} \left( p_i \times \sum_{n=1}^{X_i} t_{f_n} + (1-p_i) \times t_j \right)$$

and we have

$$E[T] = \sum_{i,j=1}^{l} \left( p_i \times \left( E[X_i] \times \frac{t_j}{2} \right) + (1-p_i) \times t_j \right)$$

The usage efficiency for both rescheduling schemes is

$$\text{Usage Efficiency} = \sum_{j=1}^{l} t_j / E[T]$$

For each job, $t_j$ are fixed, but $p_i$ vary among machines, and their values even change over the time. These $p_i$ can be obtained by failure prediction algorithms like [21, 16]. It is noteworthy that we made two very strong assumptions about the re-execution mechanism and the failure probability. In reality, when a task fails, the scheduler can assign another PI, which has a different failure probability, to take care of it. Also, even the failure probability of the same PI may change over the time. Relaxing these assumptions certainly complicates the above computation.

## 4 Usage Efficiency with Real Traces

After showing the computation of the usage efficiency in theory, in this section, we will use the real failure traces and varied workflows to derive the usage efficiency through simulation.

The failure in the simulation includes 10 two-month traces extracted from the real failure trace in [1]. It's done by randomly selecting 32 nodes in the Cluster 2 and randomly choosing two-month periods between calendar year 2001 and 2002.

We simulated the execution of 50 different random weight, communication, height, width DAGs or workflows (extracted from [13]) on 32 processing instances under 4 configurations: `No Failure`, `Recover1_without_lost`, `Recover1_lost` and `RecoverAll`. In the `No Failure`, we didn't apply failure traces. `Recover1` and `RecoverAll` are equivalent to the micro- and macro-rescheduling, respectively. After finishing its assigned task, while waiting for the new task from the scheduler, a node may fail. In this situation, the "lost" means that all data generated by the node is lost, and that finished task needs to be rescheduled. The "without_lost" means all generated data has been saved to a reliable server, and there's no need to reschedule the task once it's done.

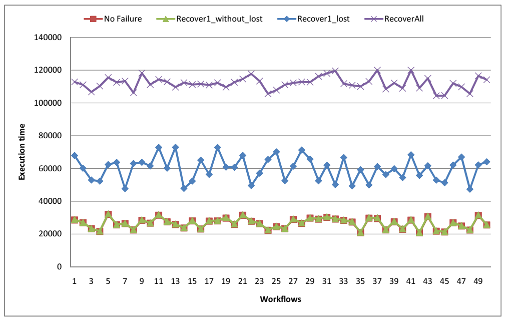

Figure 2: The performance of 50 workflows on 32 nodes.

Figure 2 shows the execution time of each workflow under 4 configurations. We only use the average of 10 failure traces in the figure for clarity. It's easy to see that the macro-rescheduling and the "lost" configurations take more time to run than the micro-rescheduling and the "without_lost" configurations. The differences between `Recover1_without_lost` and '`No Failure` are negligible.

From these results, by averaging all 50 workflows we derived that the usage efficiency of this system is 44% and 23% for the micro and macro-rescheduling, respectively. Actually, in this computation, we consider the execution time of the "No Failure" configuration as the "Required Resource". This value is different if it is computed barely from the graph of the workflow because the value from the graph doesn't account for the scheduler. In other words, it doesn't account for the resources that consumed by the nodes while waiting for the scheduler. Therefore, our computation isn't affected by the scheduler.

If we consider that the idle time spending on waiting for tasks from the scheduler is also useful (which is true in the Cloud since we have to pay for it even if we don't use it), the efficiency of the system is 100% even with failures since a node is always either busy executing a task or waiting for a task. However, with our computation it's less than 44%.

## 5 Improving Usage Efficiency in Hostile Environments

Failure is an important reason of inefficiency. System designers often apply fault-tolerant techniques in their design process such as checkpointing and temporal or spatial redundancy. However, such techniques are often the root of resource waste. Depending on the model of reaction to failures, there are different approaches to improve the efficiency.

With spatial redundancy approach, it wastes the resources in executing replicas. Hence the problem of this approach is to identify suitable number of replicas for each task [5] and their execution locations.

With temporal redundancy approach, the execution is restarted from the beginning in case of failure (macro-rescheduling). One way to improve usage efficiency is to minimize the actual execution time of the job or reduce the number of its re-execution. This can be done by choosing appropriate nodes to assign tasks [24, 17].

With checkpointing approach, the execution is restarted from the checkpoint in case of failure. It potentially creates wasted utilized resources as in the "Observation 1". To address this, one can compress checkpoints or employ process pairs as suggested in [22] or *leverage the non-volatile memory* such as phase change memory (PCM) to avoid the checkpointing overhead.

## References

[1] Los alamos national laboratory. operational data to support and enable computer science research.

[2] Data center energy forecast, 7 2008.

[3] L. A. Barroso and U. Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*, volume Lecture #6.

[4] L. A. Barroso and U. Hölzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007.

[5] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G. M. Voelker. Total recall: System support for automated availability management. In *Proc. of NSDI'04*, pages 25–25, Berkeley, CA, USA, 2004. USENIX Association.

[6] P. Bohrer, E. N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony. The case for power management in web servers. pages 261–289, 2002.

[7] D. Cappuccio. Data center efficiency beyond pue and dcie.

[8] J. P. CHRISTIAN BELADY, ANDY RAWSON and T. CADER. Green grid data center power efficiency metrics: Pue and dcie. www.thegreengrid.org, 2008.

[9] J. C. M. M. P. L. Dan Azevedo, Jon Haas. How to measure and report pue and dcie. www.thegreengrid.org, 2008.

[10] EPA. Epa report to congress on server and data center energy efficiency. Technical report, U.S. Environmental Protection Agency, 2007.

[11] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*, pages 13–23, New York, NY, USA, 2007. ACM.

[12] M. Hill. Amdahl's law in the multicore era, Jan 2010.

[13] U. H
"onig and W. Schiffmann. A comprehensive test bench for the evaluation of scheduling heuristics. In *Proc. of the 16th International Conference on Parallel and Distributed Computing and Systems (PDCS04), Cambridge, USA*, 2004.

[14] C. Lefurgy, X. Wang, and M. Ware. Server-level power control. In *Autonomic Computing, 2007. ICAC '07. Fourth International Conference on*, pages 4–4, June 2007.

[15] X. Li, Z. Li, P. Zhou, Y. Zhou, S. V. Adve, and S. Kumar. Performance-directed energy management for storage systems. *IEEE Micro*, 24(6):38–49, 2004.

[16] Y. Li and Z. Lan. Exploit failure prediction for adaptive fault-tolerance in cluster computing. *Cluster Computing and the Grid, IEEE International Symposium on*, 0:531–538, 2006.

[17] Z. Liang and W. Shi. A reputation-driven scheduler for autonomic and sustainable resource sharing in grid computing. *J. Parallel Distrib. Comput.*, 70(2):111–125, 2010.

[18] R. Paquet. Technology trends you cant afford to ignore. Gartner Webinar, December 2009.

[19] S. Park, W. Jiang, Y. Zhou, and S. Adve. Managing energy-performance tradeoffs for multithreaded applications on multiprocessor architectures. *SIGMETRICS Perform. Eval. Rev.*, 35(1):169–180, 2007.

[20] S. Rivoire, M. A. Shah, P. Ranganathan, C. Kozyrakis, and J. Meza. Models and metrics to enable energy-efficiency optimizations. *Computer*, 40:39–48, 2007.

[21] F. Salfner, M. Schieschke, and M. Malek. Predicting failures of computer systems: a case study for a telecommunication system. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 8 pp.–, April 2006.

[22] B. Schroeder and G. Gibson. Understanding failures in petascale computers. 78:012022, 2007.

[23] V. Vasudevan, J. Franklin, D. Andersen, A. Phanishayee, L. Tan, M. Kaminsky, and I. Moraru. FAWNdamentally power-efficient clusters. In *Proc. HotOS XII*, Monte Verita, Switzerland, May 2009.

[24] C. Yu, Zhifeng Wang and W. Shi. Flaw: Failure-aware workflow scheduling in high performance computing environments. Technical report mist-tr-2007-010, Waye State University, Nov 2007.