STARS

University of Central Florida
STARS

Electronic Theses and Dissertations, 2004-2019

2011

Research In High Performance And Low Power Computer Systems For Data-intensive Environment

Pengju Shang
University of Central Florida

 Part of the Computer Sciences Commons, and the Engineering Commons
Find similar works at: https://stars.library.ucf.edu/etd
University of Central Florida Libraries http://library.ucf.edu

This Doctoral Dissertation (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations, 2004-2019 by an authorized administrator of STARS. For more information, please contact STARS@ucf.edu.

STARS Citation
Shang, Pengju, "Research In High Performance And Low Power Computer Systems For Data-intensive Environment" (2011). *Electronic Theses and Dissertations, 2004-2019*. 1889.
https://stars.library.ucf.edu/etd/1889

Research in high performance and low power computer systems for Data-intensive environment

by

Pengju Shang
BS Computer Science, Jilin University, China 2005
MS Computer Science, Huazhong University of Tech. & Sci., China 2007

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the School of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Summer Term
2011

Major Professor:
Jun Wang

© 2011 Pengju Shang

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

The evolution of computer science and engineering is always motivated by the requirements for better performance, power efficiency, security, user interface (UI), etc [CM02]. The first two factors are potential tradeoffs: better performance usually requires better hardware, e.g., the CPUs with larger number of transistors, the disks with higher rotation speed; however, the increasing number of transistors on the single die or chip reveals super-linear growth in CPU power consumption [FAA08a], and the change in disk rotation speed has a quadratic effect on disk power consumption[GSK03]. We propose three new systematic approaches as shown in Figure 1.1, Transactional RAID, data-affinity-aware data placement DAFA and Modeless power management, to tackle the performance problem in Database systems, large scale clusters or cloud platforms, and the power management problem in Chip Multi Processors, respectively.

The first design, Transactional RAID (TRAID), is motivated by the fact that in recent years, more storage system applications have employed transaction processing techniques

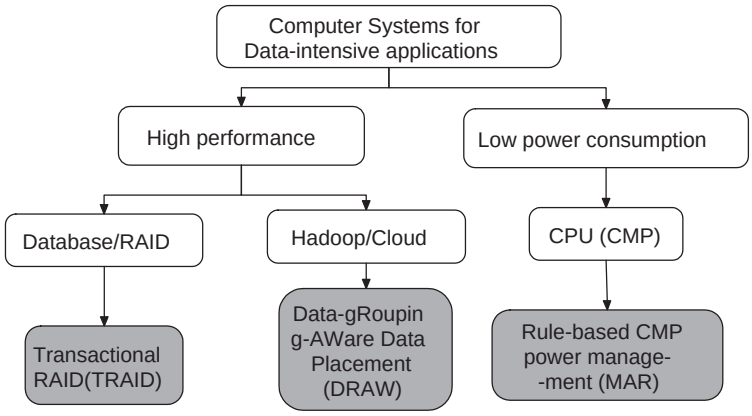


Figure 1.1: Research Work Overview

to ensure data integrity and consistency. In transaction processing systems(TPS), log is a kind of redundancy to ensure transaction ACID (atomicity, consistency, isolation, durability) properties and data recoverability. Furthermore, high reliable storage systems, such as redundant array of inexpensive disks (RAID), are widely used as the underlying storage system for Databases to guarantee system reliability and availability with high I/O performance. However, the Databases and storage systems tend to implement their independent fault tolerant mechanisms [GR93, Tho05] from their own perspectives and thereby leading to potential high overhead. We observe the overlapped redundancies between the TPS and RAID systems, and propose a novel reliable storage architecture called Transactional RAID (TRAID). TRAID deduplicates this overlap by only logging one compact version (XOR results) of recovery references for the updating data. It minimizes the amount of log content as well as the log flushing overhead, thereby boosts the overall transaction processing performance. At the same time, TRAID guarantees comparable RAID reliability, the same recovery correctness and ACID semantics of traditional transactional processing systems.

On the other hand, the emerging myriad data intensive applications place a demand for high-performance computing resources with massive storage. Academia and industry pioneers have been developing big data parallel computing frameworks and large-scale distributed file systems (DFS) widely used to facilitate the high-performance runs of data-intensive applications, such as bio-informatics [Sch09], astronomy [RSG10], and high-energy physics [LGC06]. Our recent work [SMW10] reported that data distribution in DFS can significantly affect the efficiency of data processing and hence the overall application performance. This is especially true for those with sophisticated access patterns. For example, Yahoo's Hadoop [refg] clusters employs a random data placement strategy for load balance and simplicity [reff]. This allows the MapReduce [DG08] programs to access *all* the data (without or not distinguishing interest locality) at full parallelism. Our work focuses on Hadoop systems. We observed that the data distribution is one of

the most important factors that affect the parallel programming performance. However, the default Hadoop adopts random data distribution strategy, which does not consider the data semantics, specifically, data affinity. We propose a Data-Affinity-Aware (DAFA) data placement scheme to address the above problem. DAFA builds a history data access graph to exploit the data affinity. According to the data affinity, DAFA re-organizes data to maximize the parallelism of the affinitive data, and also subjective to the overall load balance. This enables DAFA to realize the maximum number of map tasks with data-locality.

Besides the system performance, power consumption is another important concern of current computer systems. In the U.S. alone, the energy used by servers which could be saved comes to 3.17 million tons of carbon dioxide, or 580,678 cars [Kar09]. However, the goals of high performance and low energy consumption are at odds with each other. An ideal power management strategy should be able to dynamically respond to the change (either linear or nonlinear, or non-model) of workloads and system configuration without violating the performance requirement. We propose a novel power management scheme called MAR (modeless, adaptive, rule-based) in multiprocessor systems to minimize the CPU power consumption under performance constraints. By using richer feedback factors, e.g. the I/O wait, MAR is able to accurately describe the relationships among core frequencies, performance and power consumption. We adopt a modeless control model to reduce the complexity of system modeling. MAR is designed for CMP (Chip Multi Processor) systems by employing multi-input/multi-output (MIMO) theory and per-core level DVFS (Dynamic Voltage and Frequency Scaling).

1.1 Transaction Processing Systems (TPS) on Redundant Array of Independent Disks (TRAID)

In transaction processing systems, logging is the key mechanism to guarantee the durability and correctness of transactions [MHL92] [GR92] [JCM00] [FM07] and has been playing an increasingly important role in Transaction Processing Systems (TPS). Recent years have seen increasing number of I/O bound transaction processing applications, for example, in Temporal databases [MZ06] and Multidimensional databases [DPJ03][WLO01]. The log latency increases significantly in these systems because: 1) more object activities need to be logged; 2) the description of a single object includes more multi-dimensional data sources. The latency caused by logging denotes the wait time before a transaction commits: the locks on the updating data cannot be released and the transaction cannot commit until the log is flushed onto stable storage devices. This longer log latency results in committing a fewer number of transactions in a particular time frame and is becoming the bottleneck of the overall transaction processing performance.

DRAM capacity doubles every two years [OH07], so a normal OLTP database that was considered "large" can now fit into main memory easily. However, TPS always requires the log data to write to stable media before the transaction can commit. This protocol is known as write-ahead-logging (WAL). In other words, log latency is mainly determined by the amount of log data which is going to be flushed and the I/O bandwidth of the log devices.

With the increasing popularity of flash memory and flash disks, several works are proposed [LMP08, Che09, CGS09, LM07] to use these new type of storage devices for database logging. Flash devices do not suffer from the small sequential log writes (9X faster than magnetic disks) since they have no moving components. At the same time, their capacities have been increasing and the price per GB decreasing exponentially [MN06]. However, flash devices are still expensive at $5-10 per GB in 2009 (compared

to magnetic disks at $0.1-0.2 per GB), and the aforementioned researches do not solve the increasing log size issue. The solution we proposed is an ideal complement to flash device-based logging systems by decreasing almost half of the log size and therefore saving the budget on the needed capacity.

Some traditional approaches also tried to decrease the log latency. Bulk-logged [SIG06] option in SQL Server reduces the logging penalty by only recording the meta-data changes (the allocation or de-allocation information) rather than both data and metadata. It supports transaction *undo* and warm restart, but no media recovery/*redo* because there is no second-copy of the objects data in the log. Other techniques include adjusting the log file size at the database or application level, running hourly backups and truncating it nightly [JCM00]. But they do not really reduce the log latency and cost. Structuring the transaction into sub-transactions allows early partial commit of the transaction, and corresponding compensation transactions are provided for recovery purposes[KS03][ooH97]. This method improves transaction processing efficiency for independent sub-transactions but has its limitations in the dependent cases.

In our design, we propose a new **T**ransactional **RAID** (TRAID) to attack the long log latency problem for transaction processing applications. The idea is to deduplicate the information redundancy at different layers, *e.g.*, ***temporal redundancy*** (*i.e.* different versions of data copies in the time domain) on the database's log disk and ***spatial redundancy*** (*i.e.* parity redundancy or mirroring redundancy) in the RAID architecture. Our design is based on the observation that highly reliable RAID systems are widely used in commercial databases [YY01, PP01, SB03], such as RAID5 and RAID10. For these database systems, there exists an *overlap* between databases log disk and underlying storage system: the spatial redundancy provided by the disk arrays is often overlooked by the database and file system designers. On the other hand, the disk array designers are often unaware of the fault tolerant mechanisms deployed by the upper level file systems and database management systems. As a result, both groups tend to implement

an independent fault tolerant system from their own perspectives and thereby leading to potential high overhead.

TRAID exploits this overlap to improve the overall performance without violating the ACID [MH94, GR93] properties and recovery correctness of transactions. The databases using erasure coded disk arrays allow us to deduplicate redundancy by exploiting an extra XOR operation without compromising reliability. Instead of storing before and after images of the updating block, we only save the XOR result of the old parity and the new parity on the same stripe. In databases with underlying replica-based disk arrays, we only log the XOR result of old data and new data. Both of aforementioned XOR results can provide enough information for recovery when they cooperate with the parity or mirroring redundancy on disk arrays. The feasibility of the additional XOR calculation relies on the existing XOR support in RAID and rich CPU cycles.

1.2 A New Data-gRouping-AWare Data Placement Scheme for Data Intensive Applications with Interest Locality (DRAW)

With the advent of large-scale data intensive clusters, more scientists employ data parallel computing frameworks such as MapReduce and Hadoop to run their data intensive applications and conduct analyzes. In these co-located compute and storage frameworks, a task is split into many sub-tasks that execute in parallel for maximum performance. The sub-tasks are scheduled on the nodes that host the needed data, namely with the data locality to achieve better performance. Hence how to place data wisely over the clusters is crucial. Existing data parallel frameworks, *e.g.* Hadoop, or Hadoop-based clouds, distribute the data using a random placement strategy for simplicity and load balance. It is observed that, many applications have ***interest locality*** that only analyze part of a big data set. In multi-user environment such as cloud computing, hot spots

of a big data set exist when many programs share the same interests. We define a hot spot in a big data set as ***data affinity***. Unfortunately, random placement does not take data affinity into consideration. Although the overall data distribution over the cluster is balanced, affinitive data could be clustered into a small number of nodes. Subject to the per node capability constraint, many map tasks are initiated on nodes that do not host data needed and thus violate data locality.

Unfortunately in practice, many scientific and engineering applications have ***interest locality***, which means they are only interested in a *subset* of the whole data set. For example, in the bioinformatics domain, X and Y chromosomes are related to the offspring's gender. Both chromosomes are often analyzed together in generic researches rather than all the 24 human chromosomes [Dum04]. Regarding other mammal's genome data pools, the chimpanzee is usually compared with human [HL05, SLZ07]. Another example is, in the climate modeling and forecasting domain, some scientists are only interested in some specific time periods [TG09, PY08]. In summary, these co-related data have high possibility to be processed as a group by specific domain applications. Here, we formally define the "**data affinity**" to represent the possibility of two data (blocks in Hadoop) to be accessed as a group, and it is quantified as *the times that these two data have already bulk accessed*.

Unfortunately, current random placement schemes are not suit for the applications with high interest locality when only a *SUBSET* of the data is processed. This is because the affinitive data may be clustered into a small number of nodes rather than being evenly distributed because of the random-ness. To further explore why such clustered affinitive data becomes performance barriers for the MapReduce program, we need to know how a MapReduce program works. A MapReduce job is split into many map tasks to process in parallel. Map tasks intend to be allocated to the nodes with the needed data locally being stored to achieve "data locality". If the needed data is well distributed among all the nodes, map tasks can be evenly scheduled to realize ideal parallelism due to the nature

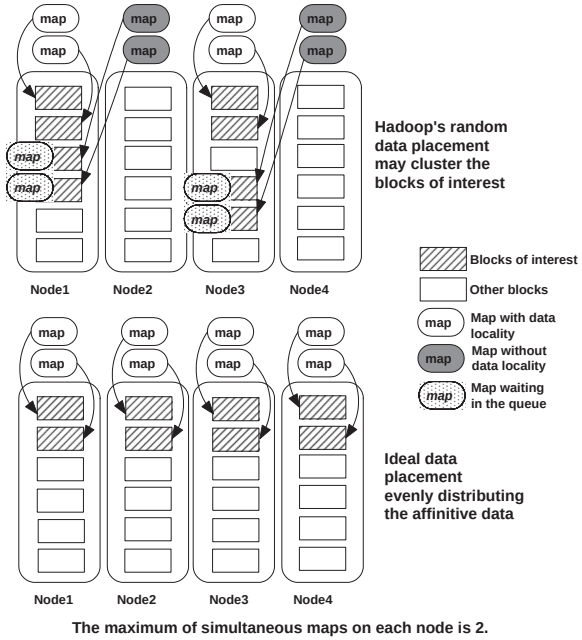


Figure 1.2: A simple case showing the efficiency of data placement for MapReduce programs.

of co-located compute and storage in data-intensive clusters. However, if the affinitive data is significantly clustered (*e.g.*, only a few nodes hold most needed data), and when the number of concurrent local map tasks per node reaches the limit on the clustered nodes (2 by default in Hadoop) [1], many map tasks are scheduled on other nodes which remotely access the needed data, or, they are scheduled on these data holding nodes but have to wait in the queue to be processed. These map tasks violate the data locality and could severely drag down the MapReduce program performance [refg]. We shown an example in Figure 4.1, where the map tasks with either remote data access or queueing delay are the performance barriers.

In this paper, we develop a new Data-gRouping-AWare data placement scheme ***(DRAW)*** that takes into account the data grouping effects to significantly improve the performance for data-intensive applications with interest locality. Without loss of generality, DRAW is designed and implemented as a Hadoop-version prototype. For a multi-rack Hadoop cluster, DRAW is launched at rack level (inter-rack) to manage the data distribution.

[1] Usually it is denoted as the maximum number of concurrent map tasks, which is determined by the hardware (processor) configurations.

DRAW consists of three components: 1) a history data access graph (HDAG) to scrutinize history data access patterns, 2) a data grouping matrix (DGM) derived from HDAG to group related data, and 3) an optimal data placement algorithm (ODPA) generating final data layout. By experimenting with real world genome indexing [refa] and astrophysics applications [refc], DRAW is able to execute up to 59.8% more local map tasks in comparison with random placement. In addition, DRAW reduces the completion time of map phase by up to 41.7%, and the MapReduce task execution time by up to 36.4% [2].

1.3 Power Management for CMP Systems in Data-intensive Environment (MARS)

Multicore processors also known as CMP have become the mainstream in the current processor market because of tremendous increase in transistor density and advances in semi-conductor technology. At the same time, the limitations in ILP (Instruction Level Parallelism) coupled with the power dissipation restrictions encourage us to enter the "CMP" era for both high performance and power saving's sake [HRV09, AAV08]. However many crucial application domains still have demand for single thread (core) performance growth [IBC06b]; and even without that, the increasing number of transistors on a single die or chip reveals super-linear growth in power consumption [FAA08b]. In this paper, performance is granted as the first priority: we try to minimize the processor's power consumption while maintaining the required performance quality.

In recent years, many power management strategies [WRW05, ICM06, WB02, XMM05, IBC06a, TT08] have been proposed for CMP processors based on DVFS(Dynamic Voltage and Frequency Scaling) [refi, ref06, ref09]. Most of these works focus on compute-intensive or memory-intensive applications, hence they try to balance the power consumption and performance based on the CPU or memory boundness of the running workloads, or the

[2]These numbers can be affected by the number of launched reduce tasks, the required data size, *etc.*.

busy/idle ratio of the CPU. This is reasonable for these non-I/O intensive applications, because the processes waiting for I/O to complete will be suspended by the operating system and the core will be given to other waiting threads [FSS07] to avoid the I/O wait time. However, in the face of today's emerging data-intensive applications, I/O wait time in processors is non-negligible [refl], and could affect the performance of CPU's power management methods, especially in the presence of many synchronized I/Os. During the CPU's I/O waiting period, the performance will be decided by the I/O subsystem rather than the CPU, hence this is a chance to lower the CPU frequency and save more power without a performance penalty.

Previous works have some limitations either in maintaining the required performance or power saving efficiency. For example, [FWB07, WMW09] focus on the non-I/O intensive cases, and their power management methods are based on CPU utilization, which is defined as $U = 100\% - (\%\ of\ time\ spent\ in\ idle\ tasks)$; the latter part is calculated by $\frac{idle\ time}{overall\ time}$. In addition, most of power management implementations are based on Linux operating systems [BMK02], in which the "iowait" field is separated from the "busy" (the sum of user, sys, and nice fields) and "idle" fields. If we apply U into data-intensive environment where the I/O operations are non-negligible, we could miss the I/O wait phases for deeper power saving. For example, assuming there are two data-intensive threads running on one CPU core as shown in Figure 1.3. Two threads have an overlapped I/O portion that would make the core enters its I/O wait phase. If we could scout both I/O wait (35%) phase and idle (10%) phase in a timely fashion, we can lower the CPU frequency to save power without sacrificing the performance. However, if we adopt the aforementioned $U = 100\% - \frac{idle\ time}{overall\ time} = 90\%$ in this case, the I/O wait phase is categorized as busy status and not used for power saving.

On the other hand, some works consider I/O factor in their power management solutions. [GFF07] divides every workload into "on-chip" part and "off-chip" part; the I/O wait time is categorized into the "off-chip" part along with the idle time, both of

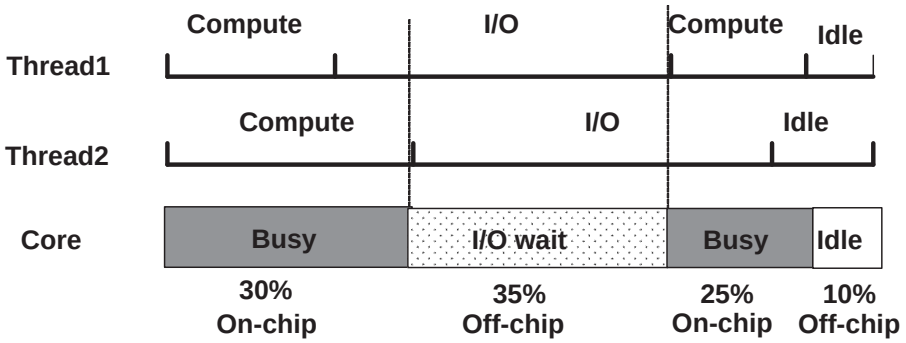


Figure 1.3: Two data-intensive threads are running on a core, the I/O wait phase and idle phase could be used for power saving

which are irrelevant to CPU's frequency. They calculate the workload characteristics as $k = \frac{onchip}{onchip+offchip}$ and scale the CPU frequency based on $\frac{k}{k+\delta} \cdot f_{max}$, where δ is the user specified performance loss and f_{max} is CPU's maximum frequency setting. Their solution works well when the workload is uniform, *e.g.*, they can scale the lowest frequency when the workload is I/O-bound ($k = 0$) or scale the highest frequency when the workload is CPU bounded. However it cannot handle the data-intensive applications where the computation and I/O operations are non-uniformly distributed. Consider the same example in Figure 1.3, we assume it takes place in one sampling period, and $k = \frac{35\%+25\%}{1} = 55\%$. If no performance loss is allowed ($\delta = 0$), [GFF07] picks the highest frequency. In fact, we could scale down a lower frequency f_{new} to save more power without compromising performance. The challenge is how to calculate f_{new} since the relationships among frequency, performance, and power consumption is too complex to be modeled when I/O factor is taken into account.

CHAPTER 2
BACKGROUND

In this chapter, we will briefly describe the relevant concepts of logging methods in transaction processing systems, and the traditional power management strategies.

2.1 Logging Methods Background

For transaction processing, the log efficiency is critical to both throughput and application response time. There are several different logging schemes to provide a balance between transactional ACID properties and the overall performance.

The basic WAL adopts *Physical Logging* [GR93] which places the old and new object states (or values) in the log record. The advantage of this method is simplicity: *undo* operation sets the page to the old value, while *redo* sets it to the new one; both *undo* and *redo* are idempotent, which is not so trivial in other logging methods. The disadvantage is large log space and long log latency: one operation, such as splitting a B-tree, may result in tens or hundreds of physical log records. In order to solve the problems of Physical Logging, *Logical Logging* [GR93] is developed. Logical Logging records the name of an *undo-redo* operation and its parameters, rather than the values. This method achieves the smallest logging space and the shortest latency; however, it may fail in media recovery. Moreover, it assumes that each action is atomic. This assumption is often violated when the action is partially complete and the system cannot decide how far to *undo*. It is a big problem in logical logging where the *undo* operation may not be idempotent. As a result, *Physiological Logging* [GR93] is developed, which uses logical logging when possible and

uses physical logging when higher reliability is required. This method transforms one complex action into a sequence of messages and page actions. These are structured as mini-transactions and log records are generated for the changes in pages or session state. In other words, Physiological Logging is a fine-grain logging method, where both the old and new values for the updating page or record are logged when necessary.

Besides the logging schemes, there are tremendous works have been done to further improve the logging efficiency. In commercial databases, e.g., SQL Server 2008, techniques like regular backup truncates old log records no longer needed for recovery to prevent the log files from consuming all of the disk space [JCM00]. Log compression [RL04] allows log records to be compressed and decompressed as they are written and read from the log files, which can provide disk usage savings. Bulk-logged [SIG06] option in SQL Server reduces the penalty of logging because the following operations are minimally logged and not fully recoverable: SELECT INTO, bulk-load operations, CREATE INDEX as well as text and image operations. Hence any-point-in-time recovery is not possible with bulk-logged option. The group commit option [YG05] accumulates several parity updates into one bigger parity update by employing journaling techniques, so that the "write penalty" for small writes can be alleviated. However, all of these logging methods and adjustments are based on the tradeoff between performance and correctness (ACID properties). Our solution in this paper exploits a new perspective to improve logging efficiency without violating ACID by utilizing the data redundancy in the storage systems.

2.2 Power Management Background

For each operational server supporting DVFS (Dynamic Voltage and Frequency Scaling), its CPU can run between a maximum frequency f_{max} (consuming the most power) and a minimum frequency f_{min} (consuming the least power).

2.2.1 CPU Frequency and Energy Consumption

As widely used in previous works, CPU power consumption is given by the following formula [10, intel]:

$$CPU\ Power = A \cdot C \cdot V^2 \cdot f \tag{2.1}$$

where A is an activity factor that accounts for how frequently gates switch, C is the total capacitance at the gate outputs, V is the voltage of the processor, and f is the operating frequency. In processors supporting DVFS, underlying circuit design inherently imposes a proportional relationship between the operating voltage and circuit frequency, which is given by $V \propto f$. As a result, equation 2.1 can be reduced to a function of processor frequency:

$$CPU\ Power = c_1 \cdot f^3 \tag{2.2}$$

Since the power consumption of all other components in the system is essentially constant and independent of the CPU frequency, we have the following simple model of the power consumption by one node in the cluster running at frequency f:

$$System\ Power = c_0 + c_1 \cdot f^3 \tag{2.3}$$

where c_0 is a constant that includes the power consumption of all components except CPU, plus the base power consumption of the CPU. Power consumption of a cluster is simply the sum of the system power consumed by its nodes, which is shown in the following equation:

$$Cluster\ Power = \sum_{i=1}^{n} c_0 + c_1 \cdot f_i^3 \tag{2.4}$$

. Equation ẽfeq:spp can be easily transformed as energy consumption as following:

$$Cluster\ Energy = c_0 \sum_{i=1}^{n} t_i + c_1 \sum_{i=1}^{n} (f_i^3 \cdot t_i) \tag{2.5}$$

where t_i is the operational time of ith node. As a result, the goal "saving the most energy" could be turned into "using the lowest CPU frequency to meet the performance requirement". 2.5, so in the following discussion, we will focus on the second adjustable part.

2.2.2 Workload partitioning

During the CPU operational period, the task being performed consists of a sequence of phases. These phases can in turn be classified as *CPU-related* (data dependency, cache hit, branch prediction instructions), *CPU-unrelated* (memory access, PCI access, I/O instructions) and *hybrid* (synchronized write, log data flush instructions). The execution time of a task is the sum of latencies to perform all phases.

If we define the response time as RT, the RT of CPU-related instructions is synchronized to the CPU internal clock and may have linear relationship with the CPU frequency. On the other hand, RT of CPU-unrelated instructions is not affected by scaling CPU frequency. For example, accesses to external devices such as SDRAM, PCI peripheral devices are synchronized to the BUS clock, which is independent of the CPU frequency. The hybrid phases consist of CPU related part as well as unrelated part. Note that these two parts may be dependent or independent to each other. For independent cases, a memory stall (or other CPU unrelated stalls) for one instruction may be completely hidden by successful completion of a later instruction [HP07]. As a result, the RT is $max\{\frac{1 \cdot p}{f_{CPU}}, \frac{1 \cdot (1-p)}{f_{BUS}}\}$. Since we are focusing on CPU frequency scal-

ing, we assume the CPU unrelated part always meets the RT requirement, so that the overall response time is only linearly determined by the CPU related part. In this way, we could transfer an independent hybrid phase as a special case of CPU-related phase. For dependent case (two parts are timesharing), the CPU related part has to wait for the CPU unrelated one to complete, for example, synchronized writes in file systems, all transaction log data flushes in write-ahead logging (WAL) based database systems. So the RT is $\frac{1 \cdot p}{f_{CPU}} + \frac{1 \cdot (1-p)}{f_{BUS}}$, Where p is the percentage of CPU related part in the hybrid region. Scaling CPU frequency will only affect the CPU related part, so the RT is not only related to CPU internal clock but also the percentage of CPU related part.

Now considering a single node with CPU frequency f_{CPU} and BUS clock frequency f_{BUS}. For a workload W, we have $W = W_{related} + W_{unrelated} + W_{hybrid}$. If T is the required time for this node to process W, then

$$T = \frac{W_{related}}{f_{CPU}} + \frac{W_{unrelated}}{f_{BUS}} + (\frac{W_{hybrid} \cdot p}{f_{CPU}} + \frac{W_{hybrid} \cdot (1-p)}{f_{BUS}}) \tag{2.6}$$

.

Now we define MAF as *the most appropriate CPU frequency*. For different type of instructions, MAF has different meanings.

1. For CPU unrelated instructions, the MAF is the minimum CPU frequency. Because the latencies will not change no matter how we scale the CPU frequency. Here notice that these CPU unrelated workloads should be large enough compared to the voltage and frequency scaling cost. For example, if the task has a large number of small $W_{CPU\ unrelated}$ which are scattered over the whole process, then the overhead of scaling CPU frequency may excess the obtained benefit.

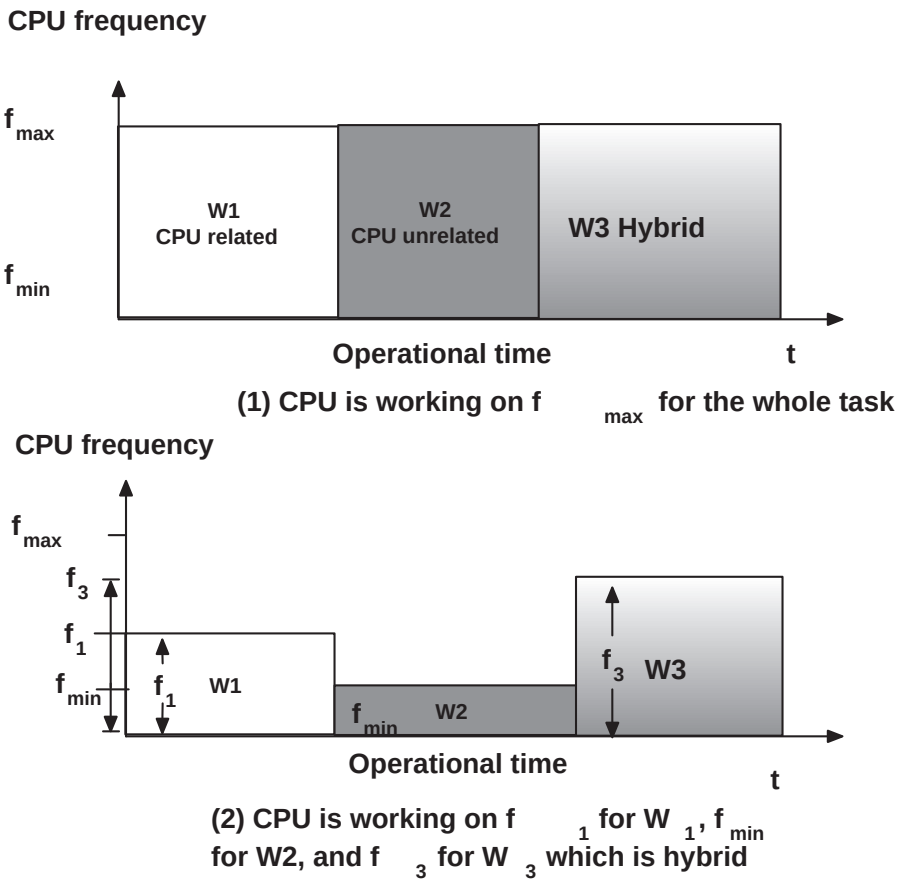


Figure 2.1: DVFS for CPU related, CPU unrelated and hybrid workloads

2. For CPU related instructions, by using *MAF*, there is no CPU idle time (CPU is fully utilized) and the SLA is just met. If we use any frequency higher (lower) than f, the response time may be unnecessarily better (worse) than SLA requirement.

3. For hybrid instructions, we can not eliminate the CPU idle time due to the CPU unrelated part. We want to find the *MAF* f for the CPU related part. By using f, the overall response time $p/f + (1-p)/f_{Bus}$ should just meet SLA requirement.

Consider a workload on a single sever which has three sub-workloads $W1, W2, W3$. $W1$ is *CPU related*; $W2$ is *CPU unrelated*; $W3$ is *Hybrid*, as shown in Figure 2.1. Figure 2.1(1) shows that the CPU works with highest frequency f_{max} for all the workloads. The length of execution times with f_{max} are t_1, t_2, t_3 for $W1, W2, W3$, respectively, and the total execution time is t. We assume the *MAF* for $W1, W2, W3$ are f_1, f_2, f_3, respec-

tively. Energy consumption in Figure 2.1(1) is:

$$c_0 \cdot t + c_1 \cdot f_{max}^3 \cdot t \tag{2.7}$$

Figure 2.1(2) shows that using MAF can achieve the same overall execution time, at the same time, based on Equation 2.5, the energy consumption on this single node in Figure 2.1(2) is:

$$c_0 \cdot t + c_1 \cdot (f_1^3 \cdot t_1 + f_{min}^3 \cdot t_2 + (f_3^3 \cdot t_3)) \tag{2.8}$$

where $t = t_1 + t_2 + t_3$.

Obviously (2.7) $>$ (2.8). Compared to f_1, f_2, f_3, any higher frequencies for those workloads will consume more energy and any lower frequencies will violate the performance requirement. As a result, Figure 2.1(2) is the ideal CPU frequency and energy management.

CHAPTER 3
TRANSACTIONAL RAID (TRAID) DESIGN

TRAID is implemented as a reliable RAID storage for TPS, and reduces log latency by minimizing log overhead. As a result, besides performance and reliability, we also show how *redo*/*undo* operations are performed correctly, *i.e.* recovery correctness and also the ACID semantics provided by relational database systems are maintained in TRAID. TRAID design exploits the overlap between the existing spatial redundancy in the most commonly used RAID architectures *e.g.* parity based (RAID5) redundancy or mirroring based (RAID10) and temporal redundancy in log disks.

3.1 Parity Redundancy: TRAID5

RAID5 is a representative storage system with parity redundancy. In database systems on RAID5, besides the original data block, there are two more copies of the same block in the system. Upon a block update request, both the before and after image of the updating block are saved to the log disk. Hence, one copy (with two versions) is on the log disk, while the other copy can be generated on the fly by using the RAID5 parity and the other blocks on the same stripe. The overlap between these two copies at any point in time enables us to log less data. More specifically, we log the XOR result of the old parity and new parity instead of before and after images of the updating block. This XOR result can provide enough references for *undo* and *redo* operations when it works with the copy in spatial redundancy. In this way, the overlap between the spatial (parity) redundancy in RAID5 and the temporal redundancy in database log

is eliminated. The XOR of successive updates technique has been successfully adopted in several recent storage system applications, such as block-level backup [YXR06] and versioning filesystem [PBA07].

Before going to the details of TRAID5 design, we recall how RAID5 processes a block *update* transaction. **(1)** Reading the target block and the parity on the same stripe from disk to memory; **(2)** Calculating the new parity; **(3)** Writing the raw data and updated data into log file for *undo* and *redo* operations. Transaction can commit after Step 3; **(4)** Writing the updated data and new parity onto disk;

The parity P in RAID5 is calculated as follows: suppose at time $T1$, we have (A_1,B_1,C_1,P_1) in RAID5, where $P_1 = A_1 \oplus B_1 \oplus C_1$; at time $T2$, one update request changes A_1 to A_2, then $P_2 = A_2 \oplus A_1 \oplus P_1 = A_2 \oplus B_1 \oplus C_1$.

In TRAID5 design, we add one new XOR result defined as TRAID-parity. Instead of logging old and new block data in Step (3), we log: $Q = P_1 \oplus P_2 = A_1 \oplus A_2$. TRAID5 architecture compared with RAID5 is shown in Figure 3.1.

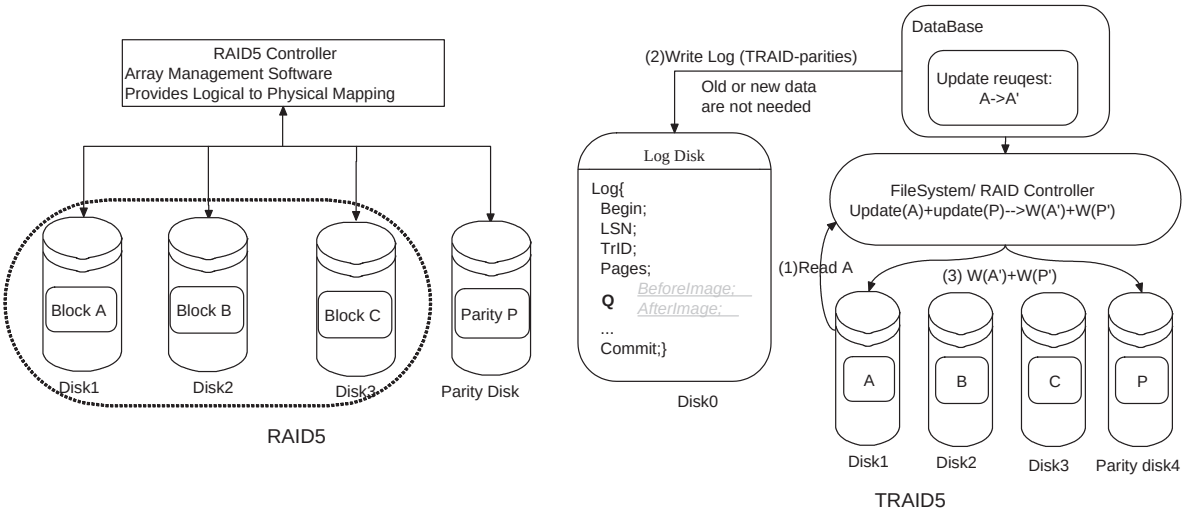


Figure 3.1: RAID5 and TRAID5

The update transaction is processed in TRAID5 in 3 steps as follow: **(1)** Reading the block and corresponding parity information from disk into memory; **(2)** Calculating the new RAID parity P and TRAID-parity Q; writing Q and all other transaction information

into the log file; the transaction can commit after Step (2); no physical *undo* or *redo* data is required; **(3)** Writing the updated block and parity P onto disk.

In this way, the log space is decreased. Because of WAL protocol, less log content in TRAID5 will decrease the log flushing time, as well as the transaction committing time.

Although, the above TRAID-parity equation is for single-block-update cases, it can be adopted for multi-block-update cases. As we know, the *write penalty* in RAID5 (the extra disk I/O to calculate the new parity) is alleviated by combining the "small writes" into one "big write". However in memory, the calculation of the new parity still requires several steps to cover all the updating blocks. For example, one update request on the stripe containing (A_1,B_1,C_1,P_1) in RAID5 may want to get a result like (A_2,B_2,C_1,P_2). The one-time write of P_2 can be $P_2 = A_2 \oplus B_2 \oplus C_1$. But before that, we will have two versions of P in the memory, such as $P_2^{'} = A_2 \oplus A_1 \oplus P_1 = A_2 \oplus B_1 \oplus C_1$ and $P_2^{''} = B_2 \oplus B_1 \oplus P_2^{'} = A_2 \oplus B_2 \oplus C_1$, where $P_2^{''}$ equals to P_2. TRAID-parity of block "A" and block "B" are obtained by taking advantage of these intermediate calculations: $Q_A = P_1 \oplus P_2^{'} = A_2 \oplus A_1$ and $Q_B = P_2^{'} \oplus P_2^{''} = B_2 \oplus B_1$. Hence, TRAID5 won't bring any extra I/O while still guaranteeing one P write as RAID5 does. For simplicity, in the following discussion about the TRAID-parity calculation, we only consider a single-block-update case.

Above equations are for the transactions updating the block only once. Given a transaction which will update the block multiple times during its life time, we need to calculate a Q list recording all the updates. However for those multiple-update transactions, the way to calculate Q will depend on whether complete rollback or partial rollback is required. We discuss these two cases in the following two sections.

3.1.1 Complete Rollback

A complete rollback means that we need to reset the database to the original state when *undo* is needed. For example, one transaction updates block A for K times from time T_1 to T_K. Complete rollback will require the system to be reset to the status at T_1, rather than other intermediate statuses. In this case, we only record the newest TRAID-parity info (at time of point T_K) Q_K for the updates on block A as follows: $Q_1 = \phi$ when $T = 1$, where ϕ denotes $NULL$; $Q_2 = P_2 \oplus P_1 \oplus Q_1$ when $T = 2$; $Q_K = P_K \oplus P_{K-1} \oplus Q_{K-1}$ when $K \geq 2$. If the old data A_1 is lost, Q_K guarantees that old data can be recovered by $A_1 = Q_K \oplus A_K$. Similarly, if the new data A_K is lost, the XOR result of Q_K and old data A_1 obtains the new data (*redo* to A_K). Table 3.1 shows the details of recovery in case of a complete rollback to A_1.

Table 3.1: TRAID-parity calculation for complete transaction rollback in TRAID5

Time	*Action*	*Parity P*	*Parity Q*	*Get A*
T(0)	*Initialize*	$P_0 = A \oplus B \oplus C$	$Q_0 = NULL$	$A = A$
T(1)	$A \rightarrow A_1$	$P_1 = A_1 \oplus A \oplus P_0$	$Q_1 = P_1 \oplus P_0 \oplus Q_0$	$A = A_1 \oplus Q_1$
T(2)	$A_1 \rightarrow A_2$	$P_2 = A_2 \oplus A_1 \oplus P_1$	$Q_2 = P_2 \oplus P_1 \oplus Q_1$	$A = A_2 \oplus Q_2$
...	*...*	*...*	*...*	*...*
T(K)	$A_{K-1} \rightarrow A_K$	$P_K = A_K \oplus A_{K-1} \oplus P_{K-1}$	$Q_K = P_K \oplus P_{K-1} \oplus Q_{K-1}$	$A = A_K \oplus Q_K$

3.1.2 Partial Rollback

In real database environment, a transaction supporting partial rollback can write to the disk several times before it commits. In this case, we need a list of Q parities i.e. $Q_1, Q_2, ..., Q_n$ for all the writes as some or all of them will be used for the partial rollback. Q for the partial rollback is calculated as follows: $Q_1 = P_1 \oplus P_0$ when $T = 1$, $Q_2 = P_2 \oplus P_1$ when $T = 2$ and $Q_T = P_T \oplus P_{T-1}$ when $T \geq 2$. If there is a system failure at a time point n with data A_n and the database needs to roll back to A_m at time point of m, where

$1 \leq m < n$, *undo* operation to A_m will work as follows: $A_m = A_n \oplus Q_n \oplus Q_{n-1} \oplus ... \oplus Q_{m+1}$. Having $Q_1, Q_2, ..., Q_n$ and A_1, we also can *redo* this transaction to any point of time m, where $1 < m \leq n$, by the following calculation. $A_m = A_1 \oplus Q_2 \oplus ... \oplus Q_m$. The details of partial recovery of data is shown in Table 3.2.

Table 3.2: TRAID-parity calculation for partial transaction rollback in TRAID5

Time	*Action*	*Parity P*	*Parity Q*	*Get any version of A*
T(0)	*Initialize*	$P_0 = A \oplus B \oplus C$	$Q_0 = NULL$	$A = A$
T(1)	$A \rightarrow A_1$	$P_1 = A_1 \oplus A \oplus P_0$	$Q_1 = P_1 \oplus P_0$	$A = A_1 \oplus Q_1$
T(2)	$A_1 \rightarrow A_2$	$P_2 = A_2 \oplus A_1 \oplus P_1$	$Q_2 = P_2 \oplus P_1$	$A = A_2 \oplus Q_2 \oplus Q_1$; $A_1 = A_2 \oplus Q_2$
...	...	...	...	...
T(K)	$A_{K-1} \rightarrow A_K$	$P_K = A_K \oplus A_{K-1} \oplus P_{K-1}$	$Q_K = P_K \oplus P_{K-1}$	$A_i = A_K \oplus Q_K \oplus ... \oplus Q_{i+1} \quad 0 \leq i < K$

It may be noted that the TRAID5 technique can be easily ported to build TRAID6 and other erasure coded arrays. The double-parity RAID or parity-based RAID6— such as RDP [CEG04]— maintains two parities P and $P^{'}$. P is same as the RAID5 parity and $P^{'}$ is used for the recovery of a second disk failure. Only P parity is used to calculate the TRAID-parity Q.

3.2 Mirroring Redundancy: TRAID10

RAID10 combines mirroring redundancy (RAID1) and striping (RAID0). Every striped block in the RAID0 part has a mirroring block in its RAID1 partner. We exploit the overlap between the temporal redundancy in the log disk and spatial redundancy in the mirroring copies to implement TRAID10.

A database running on RAID10 processes an *update* transaction in following steps: (1) Read the requested data from disks into the memory. (2) Write the Before Image (e.g. $A1$) into the log for *undo* requests. (3) Write the After Image (e.g. $A1^{'}$) into the

log for *redo* requests. Transaction cannot commit until the log is flushed due to WAL protocol. (4) Update the data anytime before or after transaction commits.

We add an XOR operation to TRAID10. Instead of logging the old data and new data, TRAID10 also records their XOR result in the log file. As shown in Figure 3.2, an update request in TRAID10 is processed as follows:

(1) Read the requested data into memory.

(2) Calculate the TRAID-parity Q based on both old version and new version data; write Q into the log file; transaction can commit after Step (2).

(3) Update the data anytime before or after transaction commits.

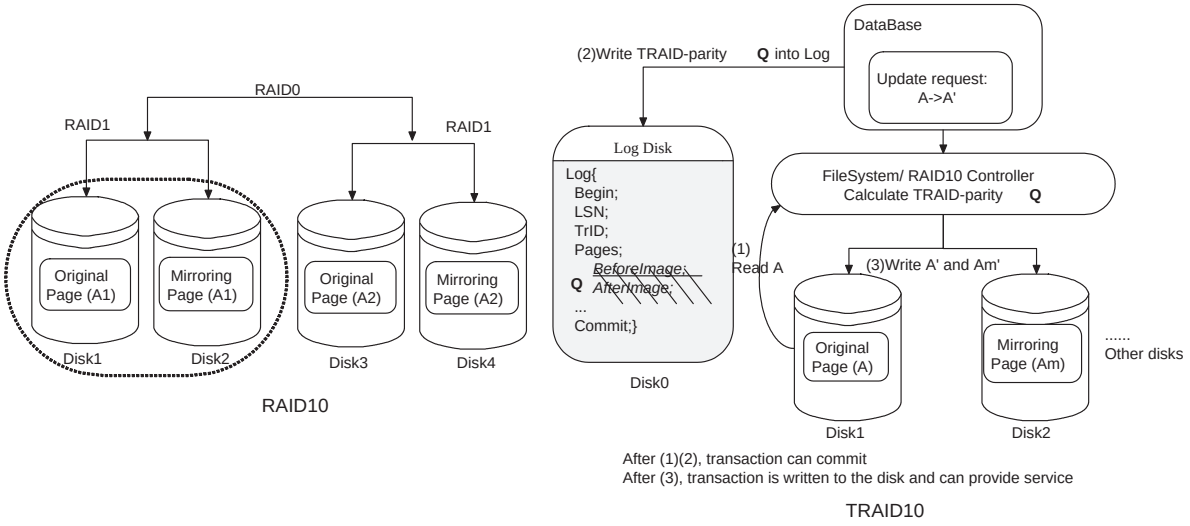


Figure 3.2: RAID10 and TRAID10

Since the mirroring redundancy from the underlying RAID10 will provide ideal reliability for either old data (before the data on disk is updated) or new data, we will always have the reference for transaction *undo* and *redo*. In this way, we utilize the overlap between the temporal redundancy and spatial redundancy to reduce log space and log flushing latency, thereby accelerate the transaction commit procedure.

The way of calculating Q in TRAID10 also depends on whether complete rollback or partial rollback is adopted, as shown in the following two sections.

3.2.1 Complete Rollback

We use the same example in TRAID5: one transaction updates block A for K times from time T_1 to T_K, and needs to be completely rollback from the current A_i where $1 \leq i \leq K$, to the original A at time T_0. We calculate the TRAID-parity Q as Table 3.3 shows.

Table 3.3: TRAID-parity calculation for complete transaction rollback in TRAID10

Time	Action	TRAID-parity Q	Get A
T(0)	Initialize	$Q_0 = NULL$	$A = A_0$
T(1)	$A \rightarrow A_1$	$Q_1 = A \oplus A_1 \oplus Q_0$	$A = A_1 \oplus Q_1$
T(2)	$A_1 \rightarrow A_2$	$Q_2 = A_1 \oplus A_2 \oplus Q_1$	$A = A_2 \oplus Q_2$
...	...	...	...
T(K)	$A_{K-1} \rightarrow A_K$	$Q_K = A_{K-1} \oplus A_K \oplus Q_{K-1}$	$A = A_K \oplus Q_K$

The "Get A" column in Table 3.3 shows how to do the complete rollback from time T_i where $1 \leq i \leq K$ to T_0. For the committed transactions, if system fails before the data on disks get updated (we have the original version of A), TRAID-parity Q can be used to get A_K by calculating $A_K = A \oplus Q_K$. Formally, Q is able to calculate A_K from any intermediate version of A, denoted as A_i where $0 \leq i < K$, by calculating $A_K = A_i \oplus Q_K$.

3.2.2 Partial Rollback

We need a list of Q parities in TRAID10 for partial rollback. Using the same example as above, we need to record every Q_i at time T_i where $1 \leq i \leq K$. The detail calculation of Q list $Q_1, Q_2, ..., Q_K$ is shown in Table 3.4.

Table 3.4: TRAID-parity calculation for partial transaction rollback in TRAID10

Time	Action	Parity Q	Get any version of A
T(0)	Initialize	$Q_0 = NULL$	$A = A_0$
T(1)	$A \rightarrow A_1$	$Q_1 = A_1 \oplus A_0$	$A = A_1 \oplus Q_1$
T(2)	$A_1 \rightarrow A_2$	$Q_2 = A_2 \oplus A_1$	$A = A_2 \oplus Q_2 \oplus Q_1$; $A_1 = A_2 \oplus Q_2$
...	...	...	...
T(K)	$A_{K-1} \rightarrow A_K$	$Q_K = A_K \oplus A_{K-1}$	$A_i = A_K \oplus Q_K \oplus ... \oplus Q_{i+1} \quad 0 <= i < K$

The "Get any version of A" column in Table 3.4 shows how to use Q and the current version of A on disk to roll the transaction backward to any point of time. If system failed after the transaction committed but before the data updated to the final version A_K, we can get A_K from the current A_i where $0 \leq i < K$ by calculating $A_K = A_i \oplus Q_{i+1} \oplus Q_{i+2} \oplus ... \oplus Q_K$.

3.2.3 Other Design Issues

Data Version Check We use TRAID-Parity Q with the updated data for both TRAID5 and TRAID10 to perform *undo*, and with the old data to do *redo*. In order to implement such recovery function, we need to know whether the data on disk is old or new. We resort to the existing ***checksum*** [KBG08, BS04, SHS01] solutions for data version check. Parity-based RAID systems always turn to checksum to detect data corruption. A checksum is a fixed-size datum computed from hash functions, fingerprints, and so forth. When the data block is being updated, a checksum is calculated based on a function f of the new data. RAID can use the checksum to tell whether the update is complete or has failed. TRAID can use this checksum to do a data version check: upon a recover request, we use the same f based on the current data on disk. If this result is as same as the checksum calculated before, it means the update operation has finished successfully, and the data on disk is new; otherwise, the data on disk is the old version.

Group Commit: TRAID can easily work with other log I/O optimization methods, such as ***group commit*** [YG05], to gain incremental performance benefit. In database with group commit, instead of starting a WAL flush immediately after a commit record is inserted, it waits for a while to give other backends a chance to finish their transactions and have them flushed by one log I/O. There are two parameters related to group commit: how many commits to wait for (commit group size), and how often (timeout) to flush the

dirty log buffer. Given a specified commit group size, since TRAID can shrink a single log record size by avoiding the before image for update and replace transactions, one log I/O in TRAID can commit even more transactions.

Adopting group commit will require the system to maintain the commit ordering and the serialization of the transactions. All the related requirements just affect the lock table and transaction table [DKO84], but have nothing to do with the log content. On the other hand, TRAID optimizes the log I/O by exploiting the overlap between the spatial and temporal redundancy so that the log content of every update transaction is reduced. As a result, TRAID complements other log I/O optimization methods, such as group commit, *etc.*.

Log space:The aforementioned discussion is based on the assumption that log is recorded in either an inexpensive disk or disk arrays, where disk I/O is expensive. If log is stored on some expensive storage devices, such as NVRAM or flash memory, where the storage space is the main concern, TRAID can reduce the log size to improve the efficiency of space utilization.

Locking granularity: Almost all database systems support fine granularity locking besides page level locking, such as record level locking (for high concurrency requirements), object level locking (for object-oriented database systems), *etc* [MH94]. For example, in Berkeley DB, page level locking is adopted for creating the database file, since a page is the basic allocation unit. While record level locking may be enabled for updates to records that are on the page, in order to achieve high concurrency. The data structures of *undo* and *redo* information in the log are affected by the locking level. The whole page (record) content is logged for page level (record level) locking, if needed [Yad07]. However, different granularity locking does not affect the recovery process. For example, in ARIES [MHL92], both page-level and record-level locking are supported in a uniform fashion in the log subsystem. In other words, no matter whether the whole page or only the record, or even only the changed attribute is logged, the recovery works

in the same way: *undo* based on the original image, *redo* based on the new image. Our design avoids the redundant original image (no matter which level locking is being used) stored in the log file when a database is working with RAID architecture.

Modularity loss: TRAID utilizes the overlapped redundancy between database log space and RAID storage space, hence these two levels are unified at some extent which will result in some loss in modularity. For TRAID5, the new generated log which contains the TRAID-parity will depend on the setting of underlying RAID system (strip size, block size, number of disks and so on); for TRAID10, the late-update version on the secondary disk will provide the *undo* reference, and the RAID controller will depend on transaction demarcations. As a result, the modularity between the storage system and application level is potentially compromised. In the future work, we want to implement an abstraction layer to eliminate this trade-off.

Data Reliability of TRAID: The core idea of TRAID is to exploit the inherent RAID redundancy to boost the performance for transaction processing systems. RAID architecture was developed to enhance the reliability of the multi-disk subsystem. Therefore, in this section we analyze the reliability of TRAID10 and TRAID5 architecture respectively, and compare them with RAID10 and RAID5 architecture. We show that reliability of TRAID5 and RAID5 are equivalent; while in TRAID10, reliability is comparable to RAID10, except during a small time frame during which it is compromised in exchange for performance.

TRAID5: The only difference between databases using TRAID5 and RAID5 is the log content, which does not affect the reliability of the storage system. Assuming that the log disk can not fail in a database system with RAID5, then more than one disk failure will result in data loss. Similarly in TRAID5, if one disk fails, the data on the failed block can be recovered by one XOR calculation. Furthermore, by using TRAID-Parity Q we can do *undo* or *redo* according to the transaction requirement. If more than one disk fails, the data will be lost since there is not enough redundancy information

to do the recovery. In other words, the TRAID-Parity Q is used to *undo* or *redo* the transactional operations, rather than doing recovery in case of disk failure. As a result, the data reliability of RAID5 and TRAID5 is same.

Let N be the number of disks in the TRAID5 and RAID5, $MTTF_{disk}$ be the mean time to failure for each disk, $MTTR$ be the mean repair time. Hence, the $MTTDL$ of TRAID5 and RAID5 are given by:

$$MTTDL_{TRAID5} = MTTDL_{RAID5} = \frac{MTTF_{disk}^2}{N(N-1) \times MTTR}$$

TRAID10: In order to calculate the reliability of TRAID10, we divide the processing of a transaction into three steps:

Step 1: Before a transaction can commit, all the transaction data and log records are in the database buffer and log buffer, respectively;

Step 2: The log records are flushed onto the log disk; transaction is ready to commit, and transaction data in the database buffer is being written to the disk;

Step 3: Transaction commits and all the transaction data and log records are on the disks.

In the step 1 and 3, TRAID10 has the same data reliability as RAID10 does because both of them have the same redundancy. In ***step 1***, the data will be lost if both database buffer and log buffer failed in TRAID10 or RAID10, as a result, the mean time to data loss (MTTDL) depends on the mean time to failure (MTTF) of the buffer modules. Let $MTTF_{buf}$ represent the mean time to failure of a buffer module, and S_{DB}, S_{LB} be the size of database buffer and the size of the log buffer respectively. The mean failure rate caused by both DB buffer and log buffer is

$$\lambda_1 = \frac{S_{DB} S_{LB} MTTR}{(MTTF_{buf})^2}$$

The MTTDL of TRAID10 and RAID10 in step 1 is therefore given by:

$$MTTDL_{TRAID10} = MTTDL_{RAID10} = \frac{1}{\lambda_1}$$

In ***step 3***, TRAID10 and RAID10 have all the data on the disks; mirrored and stripped, so the MTTDL depends on the mean time failure rate of disks. Let N be the number of disks, and $MTTF_{disk}$ be the mean time to failure of a disk. It is not straightforward to calculate the MTTDL of TRAID10 and RAID10 directly. However, we can calculate the reliability of RAID10 by using the MTTDL of RAID1 and RAID0. Suppose, we have 2-way mirroring redundancy in RAID1, the $MTTDL_{RAID1}$ is given by:

$$MTTDL_{RAID1} = 2 \times MTTF_{disk}$$

And the $MTTDL_{RAID0}$ can be denoted as (one disk failure will cause data loss):

$$MTTDL_{RAID0} = \frac{MTTF_{disk}}{N}$$

A RAID10 with N disks can be treated as a RAID0 with $\frac{N}{2}$ RAID1 groups, each of which contains 2 mirroring disks, as a result, the $MTTDL_{RAID10}$ can be given as:

$$MTTDL_{TRAID10} = MTTDL_{RAID10} = \frac{MTTF_{group}}{N/2} = \frac{2MTTF_{disk}}{N/2} = \frac{4MTTF_{disk}}{N}$$

However in ***step 2***, TRAID10 and RAID10 perform differently since we update the two mirroring copies in a different way; RAID10 writes the two copies at the same time, while TRAID10 updates one of them before the transaction commits, and then updates the other copy after the transaction commits. RAID10 in step 2 has the same data reliability as it does in step 3, since the system failure happens if and only if both the

disks in one mirroring group fail at the same time. Therefore the $MTTDL_{RAID10}$ in ***step 2*** can be also given by: $\frac{4MTTF_{disk}}{N}$.

While the situation of TRAID10 is a little more complicated. For all ***Read*** transactions, we do not need to update the data on the disks, so the $MTTDL_{TRAID10\_Read}$ in ***step 2*** is still denoted as $\frac{4MTTF_{disk}}{N}$.

For the write operations, during the asynchronous updates, the disk with un-updated copy has the old data, if this disk failed after the data on the other one is updated and before the transaction commit, we will lose the reference for possible *undo* or rollback actions. As a result, besides the normal mirroring group-failure which also happened in RAID10, we need to consider the failure of the disk containing the old data.

Before calculating the $MTTDL_{TRAID10}$ in ***step 2***, we need to formulate the write and read operations. Suppose we have totally TN transactions, the probability of write operations is P, and the average processing time for each write transaction is Tw, average processing time for each read transaction is Tr. The percentage of read operation time is:

$$\frac{Tr \times (1-p) \times TN}{(Tw \times P + Tr \times 1 - p) \times TN}$$

It reduces to

$$\frac{Tr \times (1-p)}{(Tw \times P + Tr \times 1 - p)}$$

The mean failure rate of mirroring group failure is given by

$$\lambda 2 = \frac{N/2}{2 \times MTTF_{disk}} = \frac{N}{4 \times MTTF_{disk}}$$

The mean failure rate of the disk containing un-updated data for one write request is

$$\lambda 3 = \frac{1}{MTTF_{disk} \times N/2}$$

The $MTTDL_{TRAID10\_Write}$ can be denoted as

$$MTTDL_{TRAID10\_Write} = \frac{1}{\lambda 2 + \lambda 3} = \frac{1}{\frac{N}{4 \times MTTF_{disk}} + \frac{2}{N \times MTTF_{disk}}} = \frac{4 \times N \times MTTF_{disk}}{N^2 + 8}$$

As a result, the $MTTDL_{TRAID10}$ in ***step 2*** can be given by

$$MTTDL_{TRAID10} = \frac{Tr \times (1-p)}{(Tw \times P + Tr \times (1-p))} \times \frac{4 \times MTTF_{disk}}{N}$$
$$+ \frac{Tw \times p}{(Tw \times P + Tr \times (1-p))} \times \frac{4 \times N \times MTTF_{disk}}{N^2 + 8}$$

Since the data reliability in ***step 1*** and ***step 3*** is same, we focus on one case study in ***step 2***. Suppose there is a database with an underlying RAID10, which is composed of 8 disks, the workloads are 50% read and 50% write. The MTTF for disks is assumed to be 1 million hours [SG07]. Fitting these data into the $MTTDL_{TRAID10}$ and $MTTDL_{RAID10}$, we get $MTTDL_{TRAID10} = 4.9 \times 10^5 hours$, while $MTTDL_{RAID10} = 5.0 \times 10^5 hours$, which mean 1.79% Annual Failure Rate and 1.75% Annual Failure Rate, respectively. There is 0.04% tradeoff in data reliability as compared to 40% transaction processing performance improvement. We also considered an alternative implementation of TRAID10, which can use the parity redundancy style data in logs, i.e. log XOR of old data and new data. It is anticipated that this new solution gives the same reliability as RAID10 instead of 0.04% tradeoff, but also incurs an extra overhead of XOR hardware cost that is not a part of RAID10 design.

3.3 Experimental Setup

3.3.1 Testbed

In order to implement TRAID5 and TRAID10, we modify the corresponding RAID codes in Linux kernel version 2.6.11 on a Dell Precision 690 (Intel Xeon E5345 - 2.33GHz/4.0GB RAM). 5 uniform 250G SATA disk drives with 7200 rpm rotation speed are installed. The benchmarks are TPCC and tailored TPCC(s). We create soft (T)RAID5 (with 4 disks – 3 data disks and 1 parity disk, another disk was used as log disk) and (T)RAID10 (with 4 disks, the last disk is the log disk). We launch the experiments on top of two open source database systems: Berkeley DB 4.3 version [Yad07] and PostgreSQL 8.1.4 version [refo]. There is another well-known open source database system MySQL, which uses similar logging method as Berkeley DB (Rollback Segments). We chose to hack Berkeley DB because it is more light-weight and simple. PostgreSQL is considered because it uses a different transaction logging scheme, named Multi-Version Concurrency Control (MVCC). Moreover, Berkeley DB uses page-level logging while PostgreSQL uses record level logging by default. We will evaluate our TRAID on top of these two different logging systems.

3.3.2 Implementation of TRAID5 & TRAID10

For simplicity, we only consider the transactions without partial commit, so TRAID5 and TRAID10 just need one version of TRAID_Parity.

In TRAID5, we add one TRAID5-parity calculation step using the existing XOR engine in RAID5. A hook is also added to the XOR_block function in the RAID5 source code to get the required block information and write the calculated TRAID-Parity into the buffer. When the buffer is full, or the size of group commit limitation is reached or the

database decides to write the updated transaction data to the disk, the TRAID-parities are flushed to the log disk.

In TRAID10, we create an XOR-calculation function because there is no XOR engine. One hook is added in the RAID10 controller to record two versions of the block which is to be updated. The old version and new version of the data are the inputs for XOR calculation.

3.3.3 Workloads

In order to have a fair evaluation of TRAID, we use three benchmarks: a commercial benchmark for transaction processing evaluation: TPC-C [170], and two modified versions of TPC-C as micro benchmarks.

The first benchmark, TPC-C, simulates an Online Transaction Processing (OLTP) database environment. It can measure the performance of a system which is tasked with processing numerous short business transactions concurrently. It is set in the context of a wholesale supplier operating on a number of warehouses and their associated sales districts. TPC-C incorporates five types of transactions with different complexity for online and deferred execution on a database system. These transactions perform the basic operations on databases such as inserts, deletes, updates and so on. The transactions in TPC-C and their percentage of the transaction mix are [TPC]: (1) New Order (45%): read-write. (2) Payment (43%): read-write. (3) Order Status (4%): read-only. (4) Stock Level (4%): read-only. (5) Delivery (4%): read-write.

Based on the implementation of standard TPC-C, we developed a special version of TPC-C for our test, named BTPC-C1 (Biased TPC-C benchmark1). In BTPC-C1, the key values in the queries and updates were changed from a uniformly random distribution to a biased distribution in the form of 90/10 rules. In this way, we increase the access locality so that the resulting workload is more sensitive to lock content delay, and

the log-lock content delay. By using BTPC-C1, one locked transaction can cause more transactions to wait for the lock release, so we can see how much benefit can be gained by using TRAID, i.e. a reduced log-lock latency. In the experiments with BTPC-C1, we increase the number of concurrent processes to see the performance of DB+TRAID and DB+RAID systems.

The third benchmark aims to test the performance of TRAID with a write-intensive workload, called BTPC-C2 (Biased TPC-C benchmark2). In BTPC-C2, we shield all the read-only transactions in TPC-C. Because read requests in TRAID and RAID are identical, read intensive transactions may obviate the performance improvement. Therefore, by using BTPC-C2, we can explore the advantages of TRAID for the transactions with dominant update requests.

It is necessary to consider the locking-level issue: it is known that there are page-level locking and record-level locking in database. We want to show the performance improvement of TRAID is not limited to any specific locking level. Hence the block size of a TRAID-parity is set to 512 Bytes, which is same as the default page size in Berkeley DB and PostgreSQL. In our experiments, the page size is 512 Bytes, the record size in WAREHOUSE Table is about 480 Bytes, in CUSTOMER Table is over 700 Bytes, and in STOCK Table is about 420 Bytes; while in other 5 tables, the record sizes range from 100 Bytes to 200 Bytes. 92% of the transactions will read/write the first three tables. For the majority of time, only 1 record can be fit in 1 page, which means the page level locking in our experimental configuration is comparable to record level locking.

3.4 Experimental Results

We run the TPC-C benchmark workload with the warehouse parameter set to 20, representing an initialized database size of 4 GB, which grows during each test run as new records are inserted. In our TPC-C benchmark, the input includes number of transac-

tions, number of terminals (number of concurrent processes). The output consists of the transaction processing time and the transactions per minute (tpmC). For each test, we run the given number of transactions ten times and get the average response time to analyze the TRAID performance in addition to the size of log file in each experiment. Berkeley DB and PostgreSQL are denoted as BDB and PGS for short, respectively.

3.4.1 Experiments on BDB

In Berkeley DB, the logging method is similar as the Rollback Segments-based schemes in Oracle, MySQL. These database systems simply store the modified data of each transaction in the data tables; meanwhile, the information required to rollback any particular transaction is stored in the rollback segments (in log files). BDB uses page-level logging.

3.4.1.1 Standard TPC-C Benchmark

The first experiment compares the **overall response time** of BDB+RAID and BDB+TRAID for a given number of transactions. In standard TPC-C, we set the number of concurrent processes to 10 (as the official setting), and the number of warm-up transactions to 1,000. Figure 3.3 shows the overall execution times of TPC-C on RAID10, RAID5, TRAID10 and TRAID5; Figure 3.4 shows the corresponding **throughputs**.

From Figure 3.3, we can see that compared to RAID10 and RAID5, TRAID10 and TRAID5 improves the overall response time significantly, and the improvement increases with the increasing number of transactions. The average throughput of RAID10 in Figure 3.4 is 230.21 tpm-C, while the average throughput of TRAID10 is 329.71 tpm-C, which means BDB+TRAID10 is 43.23% faster than BDB+RAID10. Similar conclusions are drawn for RAID5 and TRAID5, the average throughput of RAID5 is 197.4 tpm-C,

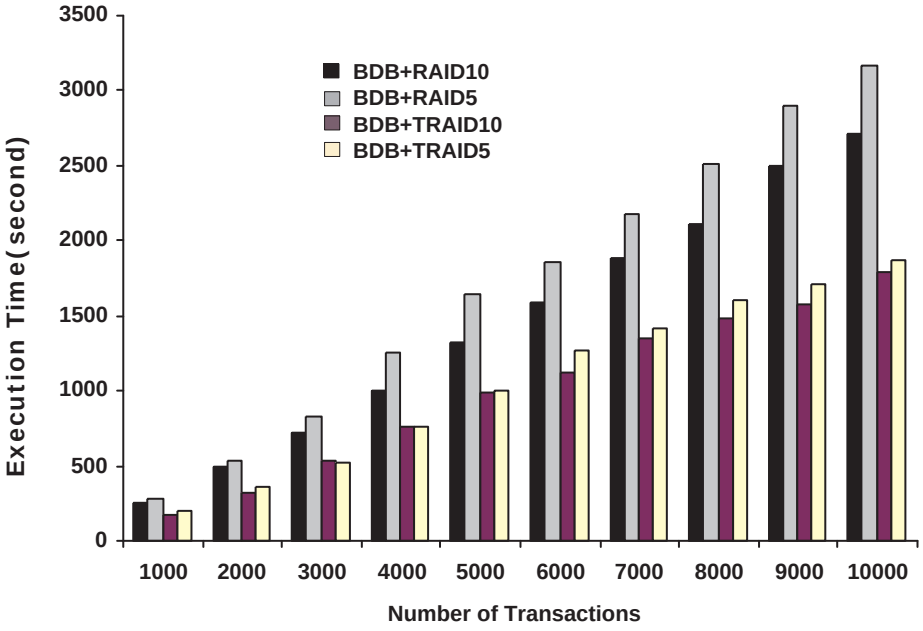


Figure 3.3: Execution Time (TPC-C)

while the one of TRAID5 is 310.5 tpm-C, which means TRAID5 outperforms RAID5 by 56.89%.

There are two reasons behind this improvement: 1) instead of writing two versions of the updating page or record into the log disk, TRAID only writes two thirds of the amount of data, which will decrease the log flushing time as well as the time when the updating transactions are locked; 2) in order to maximize the I/O bandwidth utilization, database systems usually buffer the updating writes and wait until the buffer is full to execute a big I/O. For the logging system, a similar mechanism is also used via a log buffer (group commit). Since we decreased the log content significantly, the buffer with same size for group commit in TRAID systems can hold more committing transactions compared to RAID systems, hence one same log flush I/O operation in TRAID can commit more transactions. The latter one is the key to realize the significant improvement of overall response time.

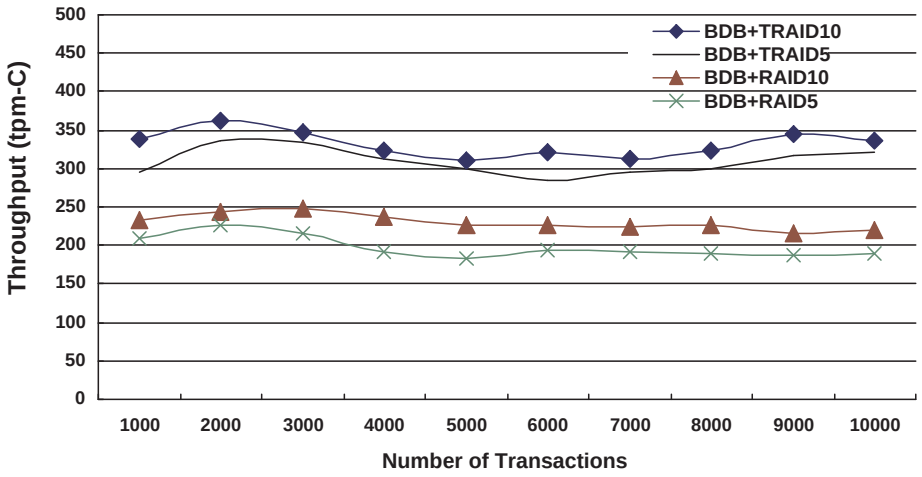


Figure 3.4: Throughput (TPC-C)

The improvement of TRAID5 over RAID5 is more significant than that of TRAID10 over RAID10, because the TRAID5-parity is the intermediate result of RAID5 parity calculation. For example, upon one request to update page A to A', the RAID5 parity calculation needs to do $P' = P \oplus A \oplus A'$ while TRAID5-parity is part of it: $Q = P \oplus P' = A \oplus A'$. As a result, we can simply read and record Q from the P' calculation rather than doing one extra calculation. However in TRAID10, we need to implement a real XOR function and do a real calculation for TRAID10-parity Q.

The throughput of TRAID5/RAID5 is a little less than TRAID10/RAID10 because we do not implement any extra optimization to eliminate the write penalty in RAID5 or TRAID5, while TPC-C has a larger percentage of writes than reads.

TRAID is also evaluated for **log size** improvement as shown in Figure 5. The size of log files in BDB+RAID10 and BDB+RAID5 is the same since all the pages being updated (including the before and after images) are logged in Berkeley DB. Both BDB+TRAID10 and BDB+TRAID5 only record the TRAID-parity information instead of before and after images. From Figure 5, we can see that TRAID10 saves the log space up to 33.7% compared to a RAID system, while TRAID5 saves 32.6%. Before analyzing this result, note that we cannot avoid recording the regular transaction information in the log file:

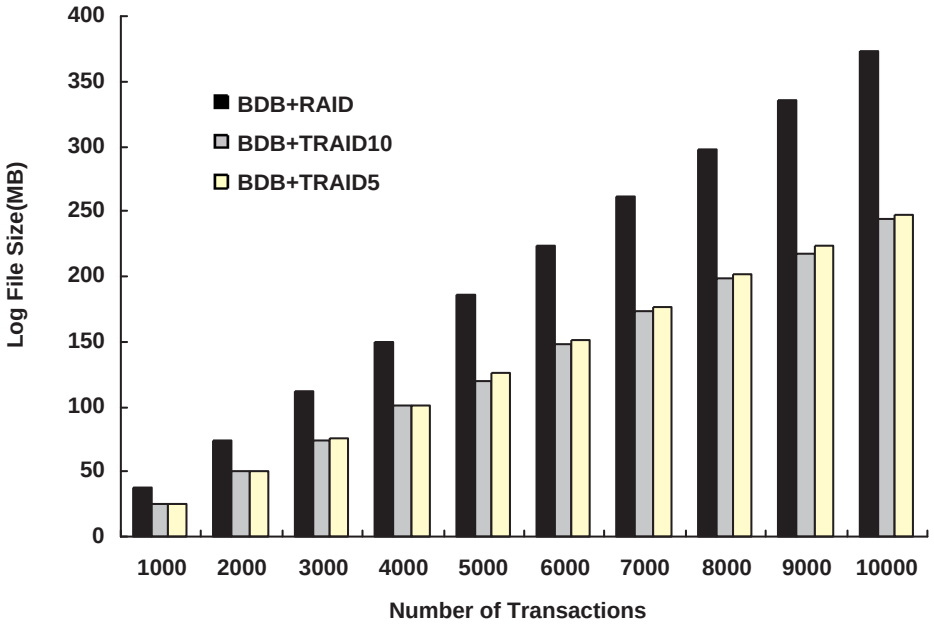


Figure 3.5: Comparison of Log Size (TPC-C)

(1) LSN, Transaction ID, *etc.*; (2) some other logged operations, such as page allocation, keep track of record counts in a B tree, mark a record on a page as deleted, *etc.*; (3) the relative large checkpoint records in BDB log file, which log all the pages are being accessed by the running transactions. Hence, by only recording the TRAID-parity in TRAID, the log size is reduced by one-third rather than 50%.

As we mentioned above, we set the parity block size as 512 Bytes in TRAID and it is the basic unit for parity computations. Actual data sizes of disk write requests (stripe size) are independent of the parity block size but are aligned with parity blocks. With this setting, one TRAID-parity will take as much space as one Before_image does in BDB log file, and it is the only way to make a fair size comparison of the TRAID5 log with Berkeley DB log. As a result, theoretically the log size of TRAID5 should be similar to TRAID10. The small difference in the experiment result is due to the different response time: TRAID5 needs a little bit more time to do all the transactions, which may result in several more checkpoint records (a new checkpoint is made every 60 seconds).

3.4.1.2 BTPC-C1 with Access Locality

The second experiment with BTPC-C1 benchmark evaluates the impact of data access locality on the TRAID performance. Since in BTPC-C1 90% of the queries and update requests focus on 10% of the data, the overall performance will be more sensitive to the log-lock-latency effect. With the increasing number of concurrent processes, the benefit of TRAID over RAID becomes more significant because TRAID reduces the wait time of subsequent transactions. We run 10,000 transactions implemented by BTPC-C1, and gradually increase the number of concurrent processes. The overall corresponding throughputs are shown in Figure 3.6.

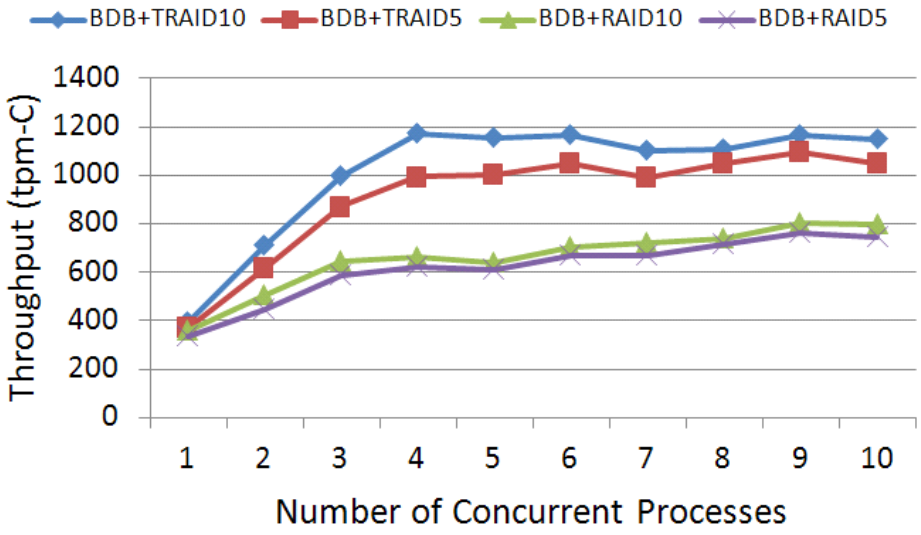


Figure 3.6: Benchmark with access locality (BTPC-C1)

From Figure 3.6, we can see the performance improvement from RAID to TRAID is not substantial when there is only 1 process. The difference between TRAID and RAID in this case (sequentially transaction processing) is the waiting-time of log-writing for sequential transactions. Also, since no concurrent transactions exist, there is no log-locking time which can further delay the transaction commit time.

The trend of throughput improvement for different number of concurrent processes is shown in the Figure 3.7. It is clear that the throughput improvement from RAID

to TRAID increases gradually with the number of concurrent transactions up till 5, and then the improvement factor starts decreasing. The lock-content delay is a crucial factor in transaction response time before the number of concurrent processes reaches 5. TRAID gains more improvement with the increasing concurrency and more lock contention because it can decrease the log-lock content delay. However, after this point, the disk I/O costs dominate the transaction response time while the lock content effect has reached the peak. The throughput improvement of TRAID over RAID becomes stable (between 41.9% to 49.6%) after the concurrency reaches a threshold of 5, where the improvement is maximized as 79.6% for TRAID10 and 63.7% for TRAID5.

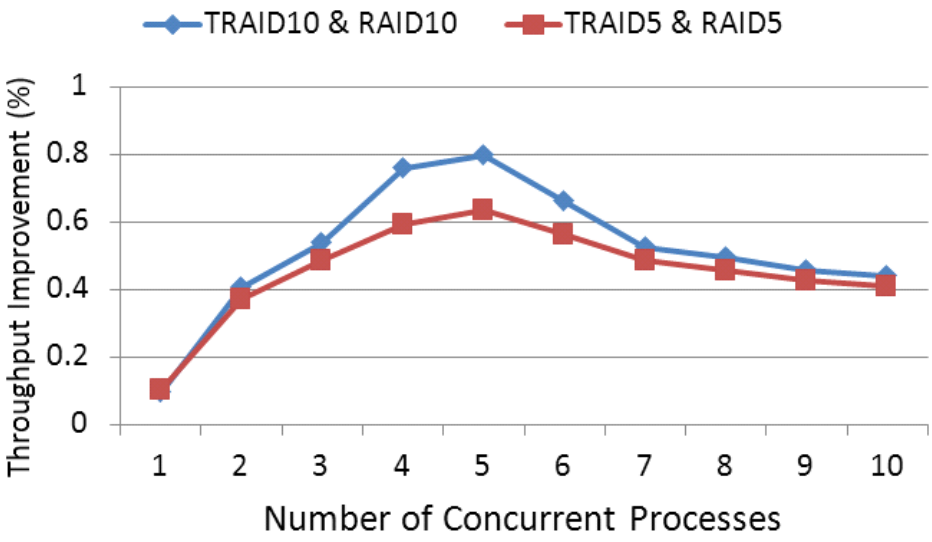


Figure 3.7: Throughput Improvement (BTPC-C1)

3.4.1.3 Write-intensive BTPC-C2

The third experiment tests the performance of TRAID for write-intensive workloads, BTPC-C2, in which every transaction needs to read and update the database. We changed the percentages of five transaction mix by deleting the read-only transactions such as Order Status transactions and Stock Level transactions, and increasing the percentages of the other three kinds of transactions. In this experiment, we set the number

of concurrent processes to 5, which is the turning point of performance improvement according to Section 3.4.1.2. The overall execution times of BDB+RAID10, BDB+RAID5, BDB+TRAID10 and BDB+TRAID5 are shown in Figure 3.8.

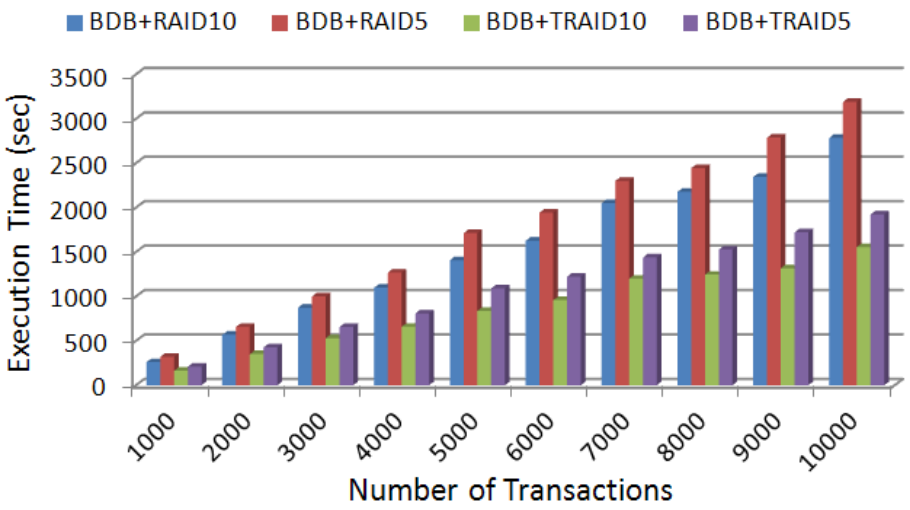


Figure 3.8: Overall Execution Time with write intensive workload (BTPC-C2)

By calculating the average improvement, TRAID10 outperforms RAID10 by 69.5% while TRAID5 outperforms RAID5 by 62.7%. Recall these numbers with standard TPC-C benmark in the first experiment, the TRAID10 and TRAID5 outperform RAID10 and RAID5 by 43.23% and 56.89% respectively. Hence TRAID obtains more improvement for writes-intensive workloads because more updates of the transactions are needed to be logged. Meanwhile, more write requests result in higher possibility of resource conflicts, which force more transactions to wait for the conflicting transaction commit. This log-locking time for a transaction commit can be reduced by TRAID. Therefore, TRAID performs even better for write-intensive workloads.

3.4.2 Experiments on PostgreSQL

PostgreSQL (PGS) uses MVCC-based logging method, which is different from Berkeley DB. Rows of a table in PGS are stored as tuples; two fields of each tuple are *xmin* and

xmax, which record the transaction ID of the transaction creating and deleting this tuple, respectively. Insertions in PGS will generate one tuple (in both database table and log files) with the *xmax* blank and the *xmin* set to the transaction ID. Deletions in PGS will find the tuple in database table and set the *xmax* field; but in sequencing log files, PGS will add a new deleting log record rather than looking up and modifying the existing tuple because log buffer operations are much faster than accessing the log files on disk. Update in PGS is no more than a concurrent insert and delete, which will write both the new tuple and the old tuple into the log files [1]. PGS uses record-level logging.

3.4.2.1 Log Size and Latency

In order to highlight the log size and logging latency, we use PGS to generate the database according to the TPC-C criterions. The database generating process includes insertions, deletions and updates, all of which will write log files. The results about log size are shown in Figure 3.9. In PGS, each log file is $16M$ by default. When it is full, a new log file is generated; there are at most 7 log files at one time; thereafter, the oldest one will be removed as the new one is added.

Figure 3.9 shows that the real data size of warehouses's tables is relatively small when compared with the log size. On average, log size is $6.01 - 6.74$ times as large as the size of generated Database. When using TRAID, the log size reduces by $28.57 - 35.48\%$ for TRAID10 and $25.37 - 31.03\%$ for TRAID5.

For the logging latency, it should be noted that PGS has two functions to perform WAL logging: XLogInsert and XLogFlush. XLogInsert is used to place a new record into the WAL buffers during the transaction processing phase; XLogFlush is made at transaction commit time to ensure that transaction records are flushed to permanent

[1] There are many ways to tune PGS's logging system to boost its performance, we only consider the most safe configuration to strictly guarantee ACID properties in our experimental settings.

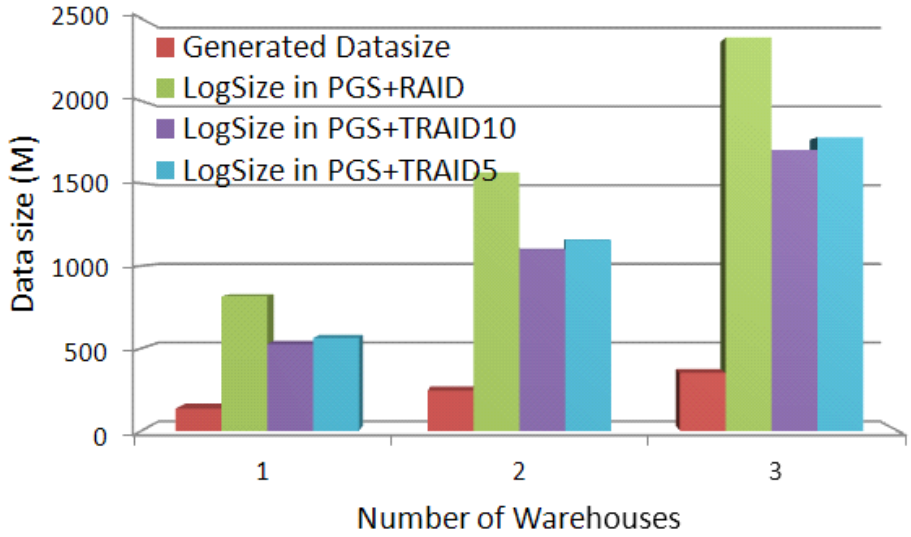


Figure 3.9: Statistics of data sizes when generating TPC-C warehouses

storage. We add a timer in XLogFlush function to quantify the log flushing latencies, as illustrated in Figure 3.10.

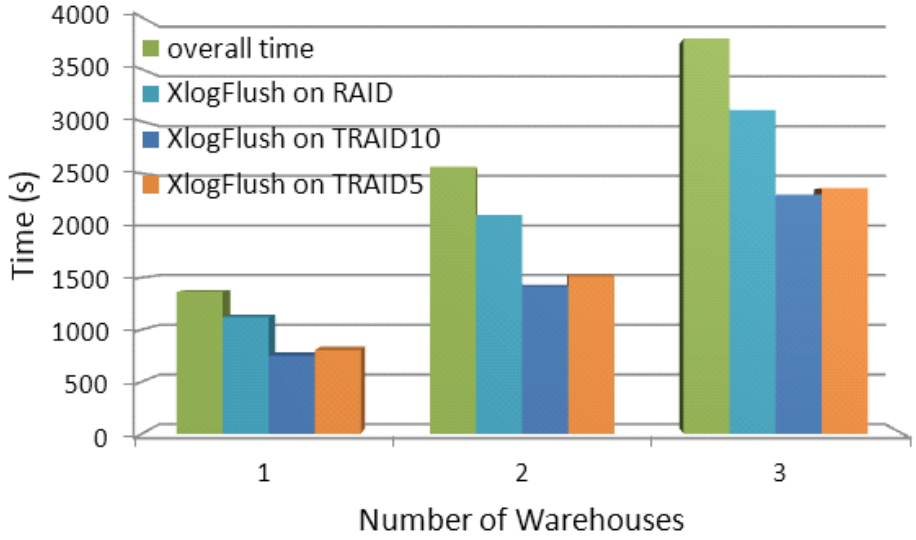


Figure 3.10: Statistics of logging latency when generating TPC-C warehouses

The results show that the logging latencies on regular RAID devices are $81.9-82.08\%$ of the overall time. When using TRAID, the above numbers become $54.9-60.5\%$ for TRAID10 and $58.9-62.2\%$ for TRAID5. On average, TRAID can reduce the log latency by $26.7-30.6\%$. Although these latencies are related to the I/O speed of the storage devices, we only focus on the improvement ratio by using TRAID.

3.4.2.2 Throughput of TRAID

We use a open-source implementation of TPC-C, named TPCC-UVA [lla04] on PostgreSQL engine to evaluate the performance of TRAID. We run the test with different numbers of warehouses and fixed 10 terminals in each warehouse. The ramp-up period is set to 20 minutes and the measurement period is set to 2 hours. The results are shown in Figure 3.11.

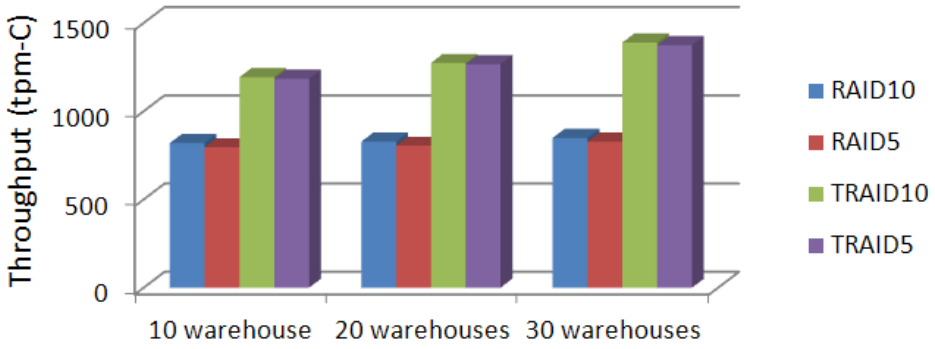


Figure 3.11: Throughput comparison of RAID and TRAID on PostgreSQL

Based on the results, TRAID10 outperforms RAID10 by 54.3%, TRAID5 outperforms RAID5 by 57.5% on average. It is interesting to observe that TRAID only reduces the log latency by $26.7 - 30.6\%$, while it can improve the overall throughput over 50%. The reason is the conflict transactions need to wait less in TRAID; a small reduction in log latency may help more transactions to proceed or commit earlier. As a result, TRAID can comparably improve the transaction processing efficiency for both page-level logging (BDB) and record-level logging (PGS).

3.4.3 TRAID & group commit

Group commit can be used in some databases to improve the log I/O efficiency. In this set of experiments, we show that TRAID combined with group commit can further improve transaction processing throughput. The database we use is Berkeley DB.

We enable group commit in BDB by setting *DB_TXN_WRITE_NOSYNC* to the database environment, which is disabled by default. If *DB_TXN_WRITE_NOSYNC* is set, Berkeley DB will write, but not synchronously flush the log on transaction commit. This means that transactions do not exhibit the durability requirement of the ACID semantics. Database integrity will be maintained. However, if the application or system fails, some number of the most recently committed transactions may be undone during recovery [BDB]. The number of transactions at risk is governed by how many log updates can fit into the log buffer, how often the operating system flushes dirty buffers to disk (controlled by a data structure called ***Timer***), and how often the log is checkpointed. We disable the checkpoint in Berkeley DB and do not set the Timer so that the log buffer will not flush to disks unless it is full. In other words, the log buffer size is exactly the same as group commit size.

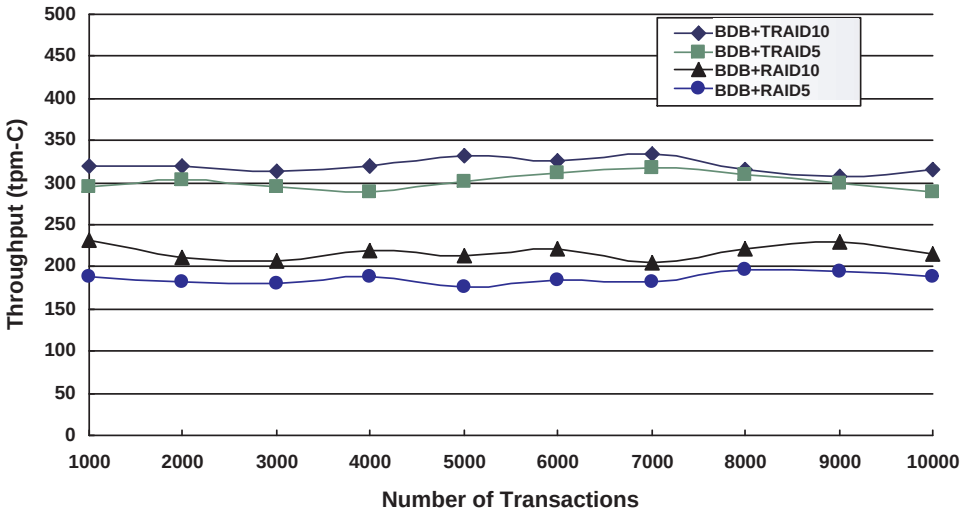


Figure 3.12: Throughput (write intensive workload BTPC-C2)

Before doing the experiment, we need to give an optimized range of log buffer size in a busy, high-DML (Data Manipulation Language) database. Common wait events related to a too-small log buffer size include high "*redo* log space requests" and a too-large log buffer may result in high "log file sync" waits. [Bur05] recommends that a value of 10MB for the log buffer is a reasonable value for Oracle Applications and it

represents a balance between concurrent programs and online users, and the value of log buffer must be a multiple of log buffer block size, 512 bytes. Following these rules 1M to 10M log buffer sizes are used in the experiment. After setting the log buffer size, we run 10,000 TPC-C transactions on top of BDB+RAID with group_commit (GC for short) and BDB+TRAID with GC. The first 200 and the last 200 transactions are used for warm up and cool down which are not counted in the overall performance. The results are shown in Figure 3.13.

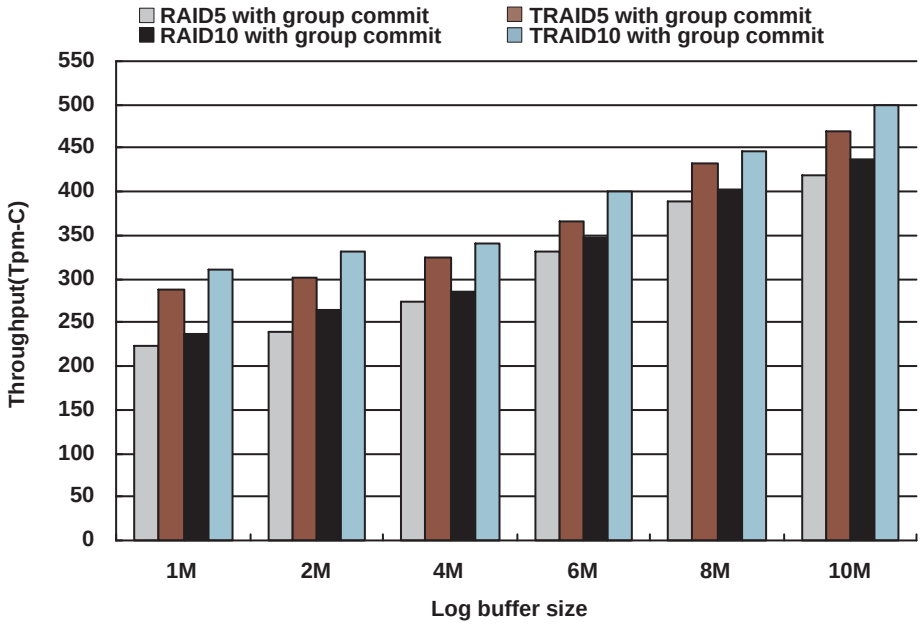


Figure 3.13: Throughput of DB with GC on RAID5, DB with GC on TRAID5

We list all the throughput improvement percentages of BDB+TRAID with GC over BDB+RAID with GC in Table 3.5. TRAID5+GC outperforms RAID5+GC by 17.8% on average, while TRAID10+GC outperforms RAID10+GC by 18.9% on average. It is interesting to observe that the throughput improvement is decreasing along with the increasing group commit size. With a larger log buffer size, more transactions to commit can fit in the buffered group; the log I/O efficiency is being improved. But the improvement is decreasing because the impact of log I/O cost is getting smaller. When log I/O latency is no longer the main bottleneck, we cannot improve the overall performance sig-

nificantly by only adopting group commit. As a result, the improvement of transaction throughput comes down to about 11% and stabilizes afterwards. The reason is that one log buffer flush (group commit) in TRAID can commit more transactions.

Table 3.5: Throughput Improvement (T5/T10 stands for TRAID5/TRAID10, GC stands for group commit, use RAID5/10 with GC as the baseline)

Log buffer	1M	2M	4M	6M	8M	10M
T5+GC	29%	26%	18%	11%	11%	12%
T10+GC	31%	25%	19%	15%	12%	12%

3.4.4 Rollback Performance

TRAID and RAID use different ways to undo transactions, so it is interesting to compare their respective rollback performance. We create transactions modifying some values of the records in CUSTOMER table, such as the customer's name, address, *etc.* All the transactions will be processed to the end but rolled back before it is committed. Only 1 process is generated to run this simulation because we just want to highlight the rollback performance. We run 500 to 3000 transactions sequentially and record the whole execution time including the transaction processing time plus the rollback time. The results are shown in Figure 3.14, the corresponding improvements are shown in Figure 3.15.

Since the transactions in this section are write-intensive, we compare Figure 3.14 with Figure 3.12. We firstly notice that RAID10 always outperforms RAID5 in non-rollback situation in Figure 3.12, but it is not always true when the transactions need to be rolled back, as shown in Figure 3.14. When the transactions are processing normally, RAID10 is doing better due to the write penalty of RAID5; while during the transaction rollback, RAID5 could do it with higher parallelism. The read from log disk will collect all the data that the undoing transaction needs, RAID5 in our experiment (consists of 4 disks) could undo the transaction on 3 disks (not including the 1 parity disk) at the same time, while RAID10 does it only on 2 mirroring groups. Our second finding is that although the

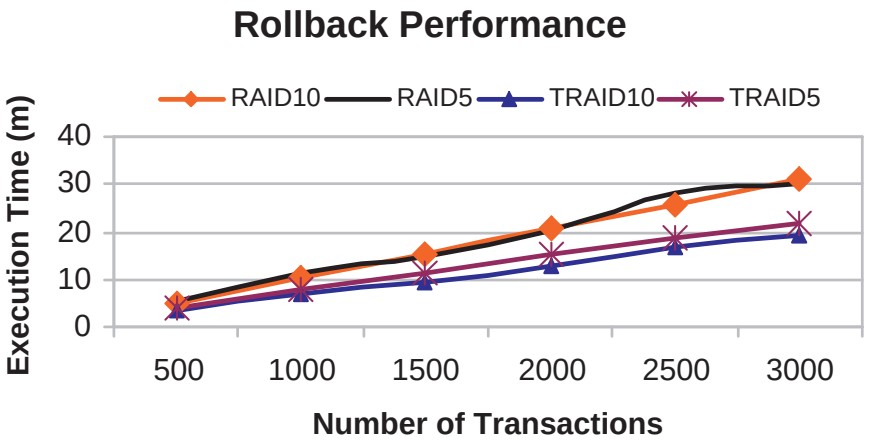


Figure 3.14: Rollback Performance

overall performance of TRAID10/5 is always better than that of RAID10/5 no matter whether there is rolled back or not, the improvement is decreased when we include the rolled back transactions.

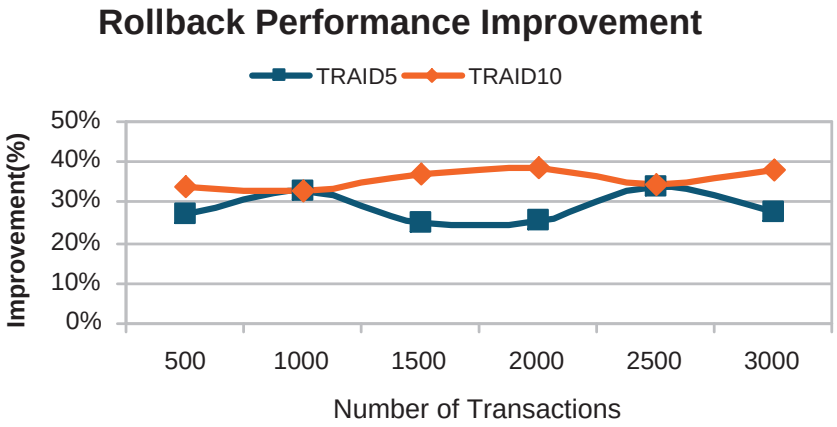


Figure 3.15: Rollback Performance Improvement

Figure 3.15 shows that when the transactions are rolled back, the average improvement of TRAID10 is 35.7%, while TRAID5 is 28.6%. These numbers in Section 3.4.1.3 are 47.4% and 61.7%, respectively. The reason is that our new rollback methods result in more disk I/O. For example, the *undo* operation on single block in RAID5 only results in one read (from log disk), one write (to the block on data disk); while in TRAID5, we will

have one read (from log disk), one read (from data disk), one XOR calculation and one write (to the block on data disk), plus the extra I/O to update the parity information on the same stripe. But this overhead is still trivial when compared with the benefit TRAID brought. The most interesting find is that when we include rollback, the overall performance of TRAID5 is decreased more than that of TRAID10 (33.1% compared with 11.7%). This is still caused by the write penalty in RAID5. When the transaction is rolled back, we write the calculated value (the old value) to the disk, this write will in turn cause extra I/O to calculate the new RAID5 parity; while in TRAID10, we introduce less overhead (one read from data disk, one XOR calculation, and one write) since there is no parity and related write penalty.

CHAPTER 4

A NEW DATA-GROUPING-AWARE DATA PLACEMENT SCHEME FOR DATA INTENSIVE APPLICATIONS WITH INTEREST LOCALITY

In this section, we design DRAW at rack-level, which optimizes the affinitive-data distribution inside the rack. There are three parts in our design: a history data access graph (HDAG) to exploit the data affinity, a data affinity matrix (DAM) to group the affinitive data, and an optimal data placement algorithm (ODPA) to generate the optimal data placement.

4.1 Motivation

The raw data obtained from the scientific simulations/sensors needs to be uploaded to the Hadoop cluster for subsequent MapReduce programs [SMW10]. In these large scale data sets, the accessing frequency and pattern of each data varies because of the applications' interest locality. For example, UCSC Genome Browser [refp] hosts the reference sequences and working draft assemblies for a large collection of genomes. It is obvious that different groups will access different subsets of these genome data: mammal [BR06], insect [HG09], or vertebrate [ZYN07]. Even in the same category, e.g. mammal, different groups may focus on different species [HL05, SLZ07].

By using Hadoop's default random data placement strategy, the overall data distribution may be balanced[1], but there is no guarantee that the data being accessed as a

[1]If the initial data distribution is not balanced, Hadoop users can start a balancer (an utility in Hadoop), to re-balance the data among the nodes.

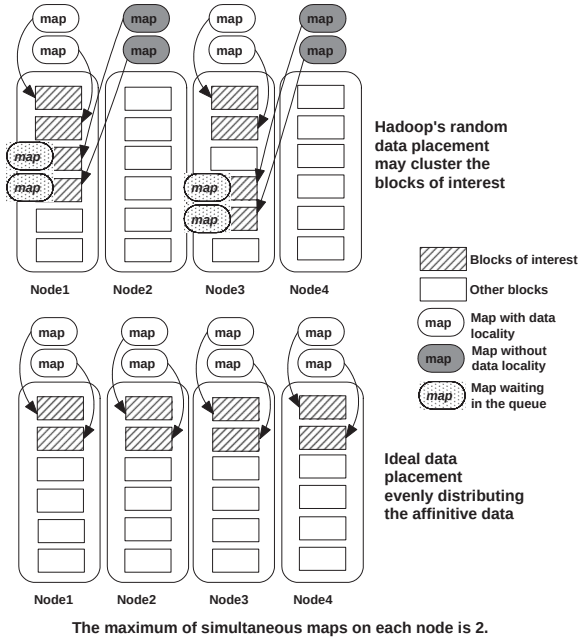


Figure 4.1: A simple case showing the efficiency of data placement for MapReduce programs.

group is evenly distributed. To further explore why such clustered grouping data becomes performance barriers for the MapReduce program, we need to know how a MapReduce program works. A MapReduce job is split into many map tasks to process in parallel. Map tasks intend to be allocated to the nodes with the needed data locally being stored to achieve "compute-storage co-locality". Without evenly distributed grouping data, some map tasks may have to be scheduled on other nodes which remotely access the needed data, or, they are scheduled on these data holding nodes but have to wait in the queue. These map tasks violate the data locality and could severely drag down the MapReduce program performance [refg]. We shown an example in Figure 4.1, if the grouping data are distributed by Hadoop's random strategy, the shaded map tasks with either remote data access or queueing delay are the performance barriers; whereas if these data are evenly distributed, the MapReduce program can avoid these barriers.

Therefore, the reason for the inefficiency of Hadoop's random data placement is because the data semantics, e.g., grouping access patterns (caused by applications' interest locality), are lost during the data distribution. On the other hand, dynamic data group-

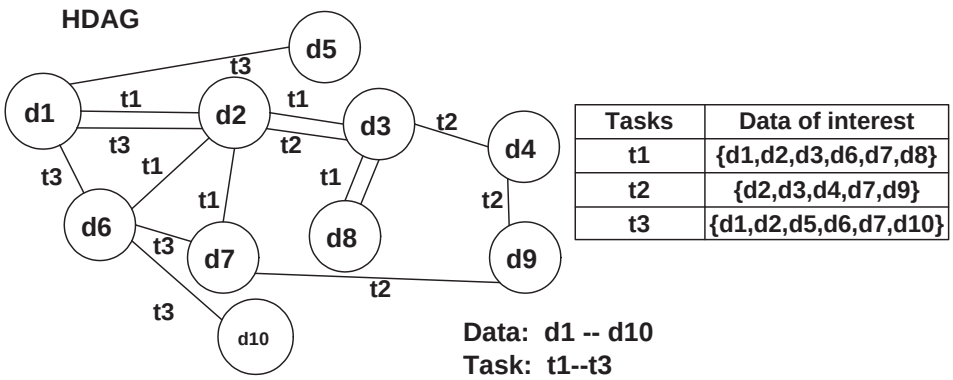
ing is an effective mechanism for exploiting the predictability of data access patterns and improving the performance of distributed file systems [**?**, **?**, **?**]. In this work, we incorporate data grouping semantics into Hadoop's data distribution policy to improve the MapReduce programs' performance.

4.2 Data-gRouping-AWare Data Placement

In this section, we design DRAW at rack-level, which optimizes the grouping-data distribution inside a rack. There are three parts in our design: a history data access graph (HDAG) to exploit system log files learning the data grouping information; a data grouping matrix (DGM) to quantify the grouping weights among the data and generate the optimized data groupings; an optimal data placement algorithm (ODPA) to form the optimal data placement.

4.2.1 History Data Access Graph (HDAG)

HDAG is a graph describing the access patterns among the files, which can be learned from data accesses history. In each Hadoop cluster rack, the *NameNode* maintains system logs recording every system operation, including the files have been accessed. A naive solution can be: filter out the files have been accessed, and every two continuously files are in the same group. This solution is simple for implementation because it only needs a traversal of the *NameNode* log files. However in practical there are two problems: first, the log files could be huge which may result in unacceptable traversal latency; second, the continuously accessed files are not necessarily related, e.g., the last file accessed by task x, and the first file accessed by task $x+1$. Therefore, we need to define checkpoint



Tasks	Data of interest
t1	{d1,d2,d3,d6,d7,d8}
t2	{d2,d3,d4,d7,d9}
t3	{d1,d2,d5,d6,d7,d10}

Figure 4.2: An example showing the History Data Access Graph (HDAG).

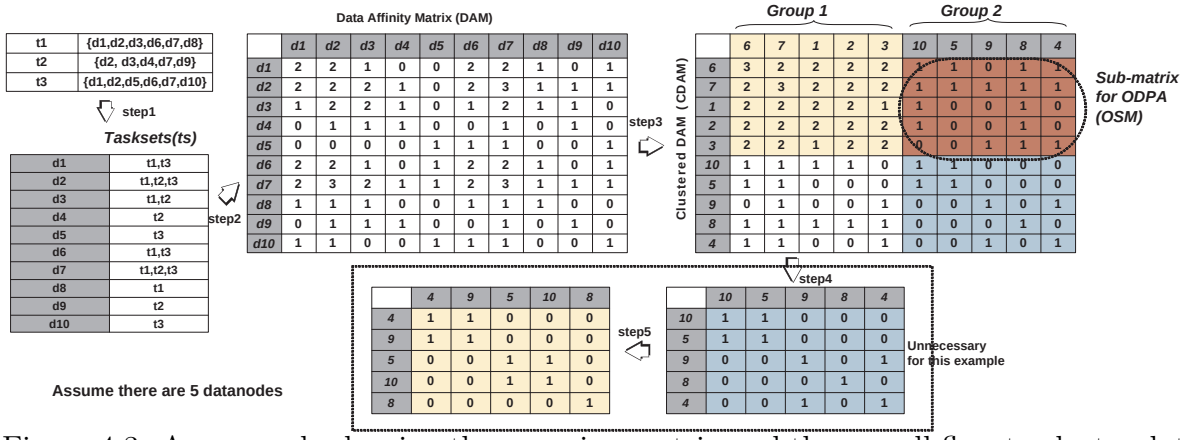


Figure 4.3: An example showing the grouping matrix and the overall flow to cluster data based on their grouping weights.

denoting how far the HDAG will traversal back in the *NameNode* logs; and we also need to exploit the mappings between tasks and files to accurately learn the file access patterns. Note that in Hadoop clusters, files are split into blocks which is the basic data distribution unit; hence we need to translate the grouping information at file level into block level. Fortunately, the mapping information between files and blocks can be found in the *NameNode*. Figure 4.2 shows an example of HDAG: given three MapReduce tasks, $t1$ accesses $d1 \sim d8$, here d is block; $t2$ accesses $d2, d3, d4, d7, d9$; and $t3$ accesses $d1, d2, d5, d6, d7, d10$. The accessing information initially generated from the log files is shown as Figure 4.2(a). Thereafter we can easily translate the table into the HDAG shown as Figure 4.2(b). This translation step makes it easier to generate the grouping Matrix for next step.

4.2.2 Data Grouping Matrix (DGM)

Based on HDAG, we can generate a data grouping matrix (DGM) showing the relation between every two data blocks. Given the same example as shown in Figure 4.2, we can build the DGM as shown in Figure 4.3 (step1 and step2), where each element $DGM_{i,j} = grouping_{i,j}$ is the grouping weight between data i and j. Every $DGM_{i,j}$ can be calculated by the counting the tasks in common between task sets of ts_i and ts_j. The elements in the diagonal of the DGM show the number of jobs that have used this data. In DRAW, DGM is a n by n matrix, where n is the number of existing blocks. As we stated before, one data belonging to group A may belong to group B at the same time; the grouping weight in the DGM denotes "how likely" one data should be grouped with another data.

After knowing the DGM in Figure 4.3, we use a matrix clustering algorithm to group the highly related data in step3. Specifically, Bond Energy Algorithm (BEA) is used to transform the DGM to the clustered data grouping matrix (CDGM). Since weighted matrix clustering problem is N-P hard, the time complexity to obtain the optimized solution is $O(n^n)$, where n is the dimension. The BEA algorithm saves the computing cost by finding the sub-optimal solution in time $O(n^2)$ [GZ99]; it has been widely utilized in distributed database systems for the vertical partition of large tables [OV99] and matrix clustering work [GZ99]. The BEA algorithm clusters the highly associated data together indicating which data should be evenly distributed. Assuming there are 5 *DataNodes* in the Hadoop cluster, the CDGM in Figure 4.3 indicates data $\{6, 7, 1, 2, 3\}$ (group 1) and $\{4, 9, 5, 10, 8\}$ (group 2) should be evenly distributed when placed on the 5 nodes. Note that we have only 10 pieces of data in our example, after knowing that $\{6, 7, 1, 2, 3\}$ should be placed as a group (horizontally), it is natural to treat the left data $\{4, 9, 5, 10, 8\}$ as another group. Hence step 4 and step 5 in Figure 4.3 are not necessary for our case, but when the number of remaining data (after recognizing the first group) is larger than the number of nodes, more clustering steps are needed.

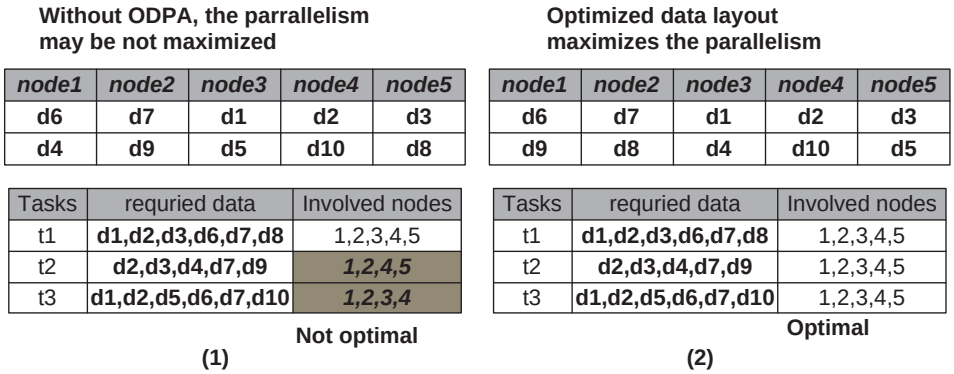


Figure 4.4: Without ODPA, the layout generated from CDGM (Clustered Data-Grouping Matrix) may be still non-optimal.

4.2.3 Optimal Data Placement Algorithm (ODPA)

Only knowing the data groups is not enough to achieve the optimal data placement. Given the same example from Figure 4.3, random placing of each group, as shown in Figure 4.4 (1), task 2 and task 3 can only run on 4 nodes rather than 5, which is not optimal.

Algorithm 4.2.1 ODPA algorithm

Input: The OSM from CDGM: $M[n'][n]$; where n' is the number of data already placed;
Output: A $n' * 2$ matrix indicating the data placement: $DP[n'][2]$;
Steps:
for each row from $M[n'][n]$ **do**
  R = the index of current row;
  Find the minimum value V in this row (not include the ones from the columns already assigned);
  Put this value and its corresponding column index C into a set ***MinSet***;
  $MinSet = C1, V1, C2, V2, ;$ // there may be more than one minimum value
  if there is only one tuple $(C1, V1)$ in $MinSet$ **then**
    //The data referred by C1 should be placed with the data referred by R on the same node;
    $DP[R][0] = R$;
    $DP[R][1] = C1$;
    Mark column $C1$ is invalid (already assigned);
    Continue;
  end if
  for each column C_i from $MinSet$ **do**
    Calculate $Sum[i] = sum(M[\star][C_i])$; // all the items in C_i column
  end for
  Choose the largest value (or the first largest value) from Sum array;
  C = the index of the chosen Sum item;
  $DP[R][0] = R$;
  $DP[R][1] = C$;
  Mark column C is invalid (already assigned);
end for

This is because the above data grouping only considers the horizontal relationships among the data in DAM, and so it is also necessary to make sure the blocks on the same

node has minimal chance to be in the same group (vertical relationships). In order to obtain this information, we propose an algorithm named Optimal Data Placement Algorithm (ODPA) to complete our DRAW design, as described in Algorithm 4.2.1. ODPA is based on sub-matrix for ODPA (OSM) from CDGM. OSM indicates the dependencies among the data already placed and the ones being placed. For example, the OSM in Figure 4.3 denotes the vertical relations between two different groups (group1:$6, 7, 1, 2, 3$ and group2:$4, 9, 5, 10, 8$).

Take the OSM from Figure 4.3 as an example, ODPA algorithm starts from the first row in OSM, whose row index is 6. Because there is only one minimum value 0 in column 9, we assign $DP[6] = \{6, 9\}$, which means data 6 and 9 should be placed on the same data node because 9 is the least relevant data to 6. When checking row 7, there are five equal minimum values, which means any of these five data is equally related on data 7. To choose the optimal candidate among these five candidates, we need to exam their dependencies to other already placed data, which is performed by the FOR loop calculating the Sum for these five columns. In our case, $Sum[8] = 5$ is the largest value; by placing 8 with 7 on the same node, we can, to the maximum extent, reduce the possibility of assigning it onto another related data block. Hence, a new tuple $\{7, 8\}$ is added to DP. After doing the same processes to the rows with index $1, 2, 3$, we have a $DP = \{\{6, 9\}, \{7, 8\}, \{1, 4\}, \{2, 10\}, \{3, 5\}\}$, indicating the data should be placed as shown in Figure 4.4 (2). Clearly, all the tasks can achieve the optimal parallelism (5) when running on the optimal data layout. With the help of ODPA, DRAW can achieve the two goals: maximize the parallel distribution of the grouping data, and balance the overall storage loads.

4.2.4 Exceptions

The cases without interest locality: DRAW is designed for the applications showing interest locality. However there are some real world applications do not have interest locality. In this case, all the data on the cluster belongs to the same group. Therefore the data grouping matrix contains the same grouping weight for each pair of data (except for the diagonal numbers); the BEA algorithm will not cluster the matrix, all the data blocks will stay on the nodes and distributed as the default random data distribution. Because all the data are equally popular, theoretically random data distribution can evenly balance them onto the nodes. In this case, DRAW has the same performance as Hadoop's random data distribution strategy.

The cases with special interest locality: The purpose of DRAW is to optimize the performance for the common applications which follow or not totally deviated from the previous interest locality. However in practice, some applications may have unpredicted access patterns that DRAW did not studied yet. These uncommon queries may suffer from bad performance because DRAW cannot guarantee these accessing data are well distributed. But this pattern will be considered into DRAW's future data organization in case it happened more times.

4.3 Analysis

In order to reveal the importance and necessity of DRAW, we need to show how inefficient the default random data distribution strategy is. Specifically, we quantify four factors in this section: the possibility for a random data distribution to be an optimal solution, the optimal degree of a given data distribution, how optimal the random data distribution can achieve, and how much improvement the random solution can achieve by using multi-replica in the same rack.

We make two assumptions: 1)uniform block size (64M) is used; 2) the default InputSplit is used, so the Hadoop block size is treated as the size for each input split [refg]. The Hadoop Map/Reduce framework spawns one map task for each InputSplit, hence we assume that the number of map tasks is the same as the number required blocks.

4.3.1 The chance that "random = optimal"

Given a cluster with N nodes, and a running application accessing M blocks that are distributed on these nodes, the "optimal data placement" should be able to distribute the M data as evenly as possible so that the corresponding M map tasks can also benefit from the maximum parallelism and data locality. However the practical Hadoop cluster's configuration may result in another "optimal" case: if the maximum number of simultaneous map tasks on each node is 2 as our assumption, and each node is equipped with a dual-core processor, then the performance of running 2 maps on a single node is the same as running 1 map. Hence we define the "***optimal data placement***" as: given a *TaskTracker* running l maps, $l \neq 0$, any other *TaskTracker* running $j \neq 0$ maps has to obey $|l - j| < 2$; any other *TaskTracker* running $j = 0$ map has to obey $|l - j| \leq 2$.

We have two cases to analyze: the number of data (M) is less than or equal to the number of nodes (N); and the M is larger than N.

Case 1: $M \leq N$ In this case, all the M blocks can be fit into one stripe on the data nodes, after which there are two ways to achieve the "optimal data placement":

1. M blocks are evenly distributed on M nodes. The possibility for Hadoop's random data placement to achieve this distribution is: C_N^M / N^M, where C_N^M means choosing M nodes from N nodes to hold the M data, N^M means the number of all possible data layouts (each block of M has N possible locations);

2. i nodes hold 1 block each, and other $\frac{M-i}{2}$ nodes are allowed to hold 2 blocks each. The possibility of this case is: $\frac{\sum_{i=1}^{\frac{M}{2}}[C_N^i \cdot C_M^i \cdot C_{N-i}^{\frac{M-i}{2}} \cdot \prod_{j=0}^{\frac{M-i}{2}} C_{M-i-2\cdot j}^2]}{N^M}$, where $C_N^i \cdot C_M^i$ means the nodes holding 1 block each, the rest of the items are for the nodes holding 2 blocks each.

Hence, when $M \leq N$, the possibility of achieving "optimal data placement" for Hadoop's random data placement is the combination of above two euqations:

$$\frac{C_N^M + \sum_{i=1}^{\frac{M}{2}}[C_N^i \cdot C_M^i \cdot C_{N-i}^{\frac{M-i}{2}} \cdot \prod_{j=0}^{\frac{M-i}{2}} C_{M-i-2\cdot j}^2]}{N^M} \tag{4.1}$$

Case 2: $M > N$ In this case, $M = kN + d = (k+1) \cdot d + k \cdot (N-d)$, where $k \geq 1$, $d \geq 0$. The "optimal data placement" can be achieved by distributing the blocks in two groups: the first group has d nodes, each of which host $k+1$ blocks; the second group has $N-d$ nodes, each of which hosts k blocks. In this way, each node will be assigned the same number of map tasks. For random data placement, the possibility of achieving this is shown in Equation 4.2. The number of all possible data layout is still N^M.

$$\frac{C_N^d \prod_{j=0}^{d-1} C_{M-(k+1)j}^{k+1} \cdot C_{N-d}^{N-d} \prod_{j=0}^{N-d-1} C_{M-(k+1)d-k\cdot j}^{k}}{N^M} \tag{4.2}$$

Hence, the Possibility of achieving the "OPtimal data placement" (POP) for Hadoop's default data placement algorithm is the combination of Equation 4.1 and Equation 4.2. It is clear that POP is related to three factors: the number of data(blocks) of interest, the number of nodes in the Hadoop cluster, and the maximum number of simultaneous map tasks on a single node. We already assume the last factor as 2 in this paper. We plot the trajectory of POP in Figure 4.5. Note that in the z axis, we show the *log* value of the POPs for clarity: when $z = 0$, it means the random data placement is the "optimal data placement"; when $z < 0$, it means the possibility is 10^z. As Figure 4.5 shows, for a specific number of data of interest (> 2), along with the increasing number of nodes in the

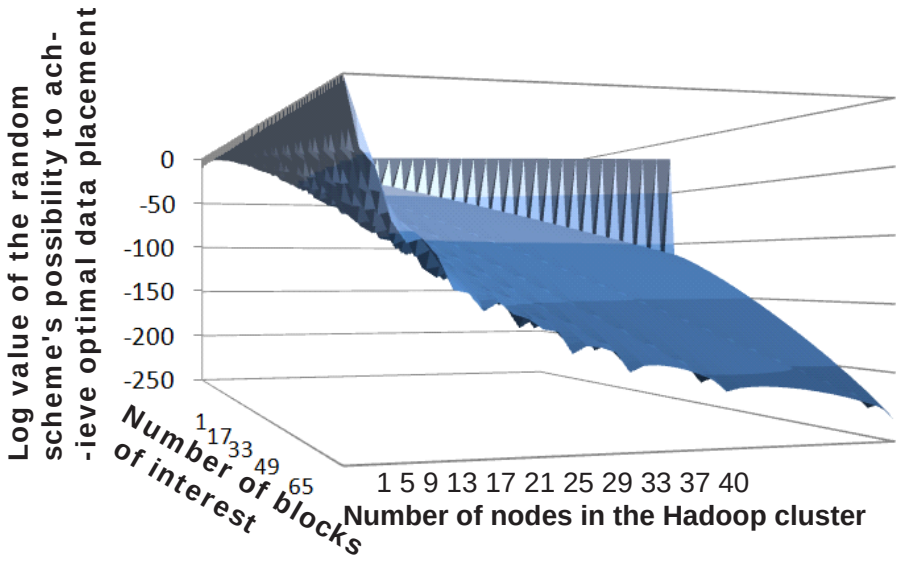


Figure 4.5: The Possibility of achieving "OPtimal data placement" (POP) for Hadoop's default data placement algorithm.

Hadoop cluster, POP is decreasing; given a cluster with a specific number of nodes, the increasing number of data of interest leads to a lower POP as well. Based on our analysis, for a small scale cluster as our test bed which only has 40 nodes, when the number of data of interest is larger than 5 ($320M$), it is highly unlikely that ($POP = 10^{-100}$) the random data placement to achieve optimal data layout. Unfortunately, most of data-intensive applications work on large-scale (GB or even PB) data [DG08].

4.3.2 The optimal degree of a given data distribution

As we already proposed the definition of the optimal data distribution, the ones do not satisfied the requirement are not optimal, but it is still interesting to know "how optimal" they are. Therefore, we propose a concept "optimal degree of data distribution" , denoted as ***Degree***. *Degree* is between $[0, 1]$: *Degree* for the "optimal data placement" in Section 4.2.3 is 1; in the cases when all the interested data are clustered in one node, *Degree* is 0.

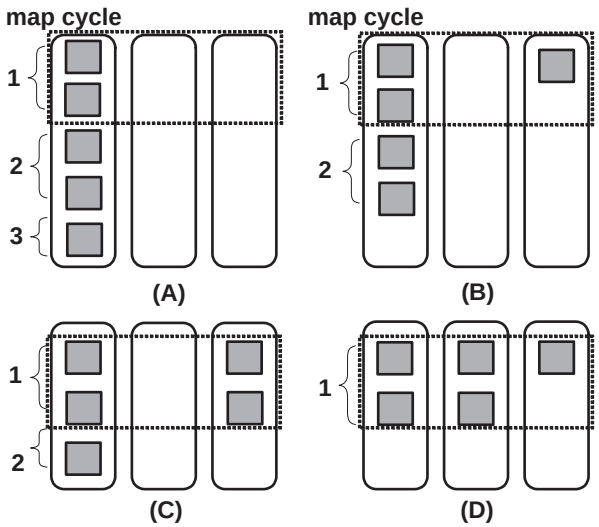


Figure 4.6: An example to show how to use Equation 4.4 to calculate the optimal degree of data distribution: $Degree(A) = 0$(clustered), $Degree(B) = Degree(C) = 0.5$(suboptimal), $Degree(D) = 1$(optimal).

To calculate $Degree$, we assume there are N nodes, M data of interest, the maximum number of simultaneous map tasks on a single node is k, the number of data of interest on i_{th} node is B_i, so $M = \sum_{i=1}^{n} B_i$. As a result, the $Degree$ can be defined as Equation 4.3. The $max(B_i) - B_{opt}$ means the difference between the node storing the max number of (interest) data in a random distribution and any node in optimal data distribution; the less this number is, the more efficient the random solution is; B_{opt} can be denoted as $\lceil \frac{M}{N \cdot K} \rceil \cdot k$. Symbol "$\lceil \rceil$" is used because of simultaneous running map tasks ($x \sim x + k - 1$ blocks result in the same number of map cycles to run the maps simultaneously); note that the ks cannot be canceled because of the existence of "$\lceil \rceil$".

$$\begin{aligned} Degree &= 1 - \frac{\lceil (max(B_i) - B_{opt})/k \rceil}{\lceil (M - B_{opt})/k \rceil} \\ &= 1 - \frac{\lceil (max(B_i) - \lceil \frac{M}{N \cdot K} \rceil \cdot k)/k \rceil}{\lceil (M - \lceil \frac{M}{N \cdot K} \rceil \cdot k)/k \rceil} \end{aligned} \tag{4.3}$$

We use an example in Figure 4.6 to show how to use Equation 4.3. Assume we have $N = 3$ nodes, $M = 5$ data of interest, and $k = 2$ as in previous analysis, Figure 4.6 shows four different data distribution. $B_{opt} = \lceil \frac{M}{N \cdot K} \rceil \cdot k = 2$, hence in optimal data distribution,

the maximum number of blocks on a single node is 2. In practical MapReduce running, **(A)** can finish the five maps on the five blocks in three mapping cycles (because $k = 2$), while **(B) and (C)** need two cycles, **D** needs only one cycle. We can calculate the *Degree*s for these four cases to quantify their efficiency: **(A)**, $max(B_i) = M = 5$, hence the $Degree(A) = 0$, which means (A) is the least optimal distribution; similarly we also get $Degree(B) = 0.5$, $Degree(C) = 0.5$ (suboptimal) and $Degree(D) = 1$ (optimal).

4.3.3 The "optimal-degree" of the random distribution

We already proved random distribution can hardly achieve optimal solution in Section 4.3.1, but it is also necessary to show how close the random and optimal data distributions are. Therefore we quantify the level of approximation (LoA) between random and optimal solutions as shown in Equation 4.4 [2]; where $P(Degree)$ means the possibility of random solution achieves the distributions with the $Degree$ of optimal data distribution, e.g., $P(0)$ is the possibility for random data distribution to cluster all the data of interest onto the same node ($Degree = 0$).

$$\begin{aligned} LoA &= \int_{Degree=0}^{1} Degree \cdot P(Degree) \\ &= \int_{maxB_i=M}^{\lceil \frac{M}{N \cdot K} \rceil \cdot k} Degree \cdot P(Degree) \end{aligned} \tag{4.4}$$

It is observed that LoA is a function related to three factors: M (number of blocks of interest),N (number of nodes in the cluster),and k (number of allowed simultaneous map tasks on a single node). We use sampling technique to obtain the trajectories of LoA to learn how the factors affect the efficiency of random data distribution. We set $N = 40$ in the simulation according to the cluster size of our test bed; $M = 10, 30, 60, 80, k = 1, 2$.

[2] In other words, LoA denotes how sub-optimal the random distribution is, on average. The more LoA is close to 1, the closer the random and optimal approaches are.

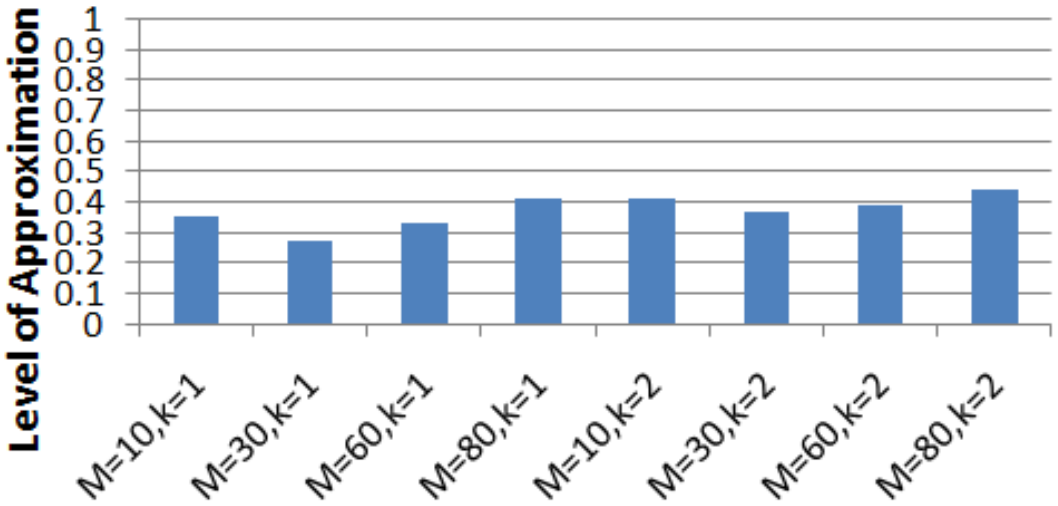


Figure 4.7: Level of approximation between random data distributions and the optimal solution, the number of nodes N is set to 40.

The results are shown in Figure 4.7. Larger k always increases LoA because the more simultaneously running map tasks will hide the unbalanced data distribution better; M, the number of data of interest, affects LoA in an uncertain way: when $M << N$ (M=10,N=40), increasing M may decrease LoA but when M is close the N or $M > N$, increasing M leads to a larger LoA. However, the average LoA for $k \leq 2$ is less than 45%, which means the random data distribution can only achieve "less-than-half-optimal" data distribution, on average.

4.3.4 Multi-replica per rack

In previous analysis, we assume that there is only one copy of each data existing in each rack. This assumption is derived from the practical Hadoop configurations, e.g., Hadoop with single-replica for each data [refn, ZZL09], Hadoop with three replica for each data but put into three different racks [refd], *etc.*. However, there are some Hadoop clusters keep two or even three copies of the same data in the same rack [refg] to provide better write performance. As we stated in Section 4.1, the more replica for each data in the

same rack, the more optimal data distribution the random strategy can achieve (given that any two replica cannot stay in the same node). In order to prove our DRAW is still necessary for multiple replica Hadoops, we launch intensive experiments as sensitivity study in Section 4.5.3.

4.4 Methodology

4.4.1 Test Bed and Applications

Our test bed consists of 40 heterogenous nodes in total with Hadoop 0.20.1 installed on it. All these nodes are in a single rack. In our setup, the cluster's master node is used as the *NameNode* and *JobTracker*, whereas the 39 worker nodes are configured to be *DataNodes* and *TaskTrackers*. The cluster and node configurations are omitted due to space limitation.

We launched two applications on the real scientific data in our experiments: one from bio-informatics area, and one from astrophysics research.

Bowtie [refa], is a real application from genome research. This application indexes the chromosomes with a Burrows-Wheeler indexing [refb] algorithm to keep their memory footprint small. The genome's indexing is a strategy for rapid gene search or alignment. In our experiments, Bowtie's indexing algorithm is implemented in MapReduce framework. The data is about $40GB$ genome data that is downloaded from [refp], including human, horse, chimpanzee, etc. 32 species in total. The application is performed on specific species, or random combinations of species (interest locality).

The second application is a mass analyzer working with astrophysics data sets for halo finding [refc]. The data sets are comprised of particle positions and velocities. Specifically, each particle has one corresponding file, which has the following content: $position(x, y, z)$, $velocity(V_x, V_y, V_z)$, $particlemass$, and $particletag$. The total size of the download is about $10GB$ of particle data in total. And each particle file is exactly $512MB$. The mass analyzer reads the mass data for specific particles, or combinations of particles, and calculates the average mass in each area (interest locality); the area size is pre-defined.

We first run each application 20 times on randomly chosen data sets to build the grouping history. Then DRAW is used to re-organize the data. Finally we re-run the applications on the newly distributed data, and compare the performance of the applications running on DRAW data and the randomly placed data, respectively.

4.4.2 Implementation

Data grouping learning: Data grouping information can be derived from the *NameNode* log file, which maintains all the system operations. We filter out the file accessing information from the log file first, and the files accessed by the same task (denoted by the same "JobID") are considered as grouping files in HDAG, as shown in Figure 4.2. After the log traversal, a matrix showing the data grouping at file-level (file-grouping) can be generated. The mapping between filenames and blocks is exploited [3] to generate the "Data Grouping Matrix (DGM)" at block level (as shown in Figure 4.3). In order to improve the log learning efficiency, we set a check point using the time stamp when we used DRAW last time, thus the current log learning starts from the most recent operations back to the check point.

[3]By using Hadoop system call "fsck" with parameters "-files -blocks -locations" for each file.

Data grouping clustering: Given the data grouping matrix, Bond Energy Algorithm is used to perform matrix clustering. The size of each group is same as the number of nodes in the cluster. In this way, all the data groups should be placed one after another from right-top to the left-bottom in the clustered DGM(CDGM) (Figure 4.3). As we explained in Section 4.2.3, in order to achieve the optimal data placement, we also need ODPA algorithm to generate the final DRAW matrix showing the target data layout.

Data placement: The most challenging part of this work is how to implement the data re-organization according to the "optimal data layout" generated by DRAW. In a Hadoop cluster, all the information about the block locations, and mappings between the files and blocks, are located in the *NameNode*. If we want to re-organize the data in the cluster, we need to, accordingly, modify the information in the *NameNode*. However, the *NameNode* does not provide any functionality that allows the users to modify this information; it just passively updates them based on the periodical reports from the living *DataNodes*. On the other hand, the *DataNodes* only support read, write, and delete operations, but there is no available function to migrate the data among the *DataNodes*. We solve this problem by modifying the Hadoop storage system. Our observations show that each block and its metadata on the *DataNode* are registered in a log file, which reports to the *NameNode* for updating. By logging in each *DataNode* which requires data re-organization, we migrate the data, metadata, and its registration information as a group. After the migration, we temporarily change the heartbeat interval of the Hadoop cluster so that the most-up-to-date data layout can be updated in the *NameNode* as fast as possible. DRAW is implemented as an off-line tool for Hadoop cluster, users need to manually launch it to achieve the "optimal data layout" based on the latest data grouping information.

4.5 Experimental Results and Analysis

In this section, we present four sets of results: the unbalanced data distribution caused by Hadoop's default random data placement; comparison of the traces of the MapReduce programs on the randomly placed data, and the DRAW's re-organized data; the sensitivity study used to measure the impact of the NR (number of replica for each data block in Hadoop) on DRAW; and the overhead of performing DRAW data re-organization.

4.5.1 The Data Distribution

Intuitively, the data distribution may be related to the way the data is uploaded. There are two ways for the users to upload data: bulkily upload all the data at once; or upload the data based on their categories, e.g. species or particles in our cases. The second way considers the human-readable data grouping information (in our case, data belonging to the same species or particles are assumed to be highly related) rather than the blindly uploading as in the first method. We upload the data to our test bed by using these two data uploading methods, 20 times for each. The overall data distributions are similar in these runs.

First, after bulk uploading the genome data of six species (a subset of our $40GB$ genome data), the data distribution (from a randomly picked run) is shown in Figure 4.8 (1). Given a research group only interested in human [HL05, SLZ07], the requiring data is clustered as shown in Figure 4.8 (2). The human data is distributed on only half (51.3%) of the cluster, which means the parallelism for the future MapReduce job is not optimal.

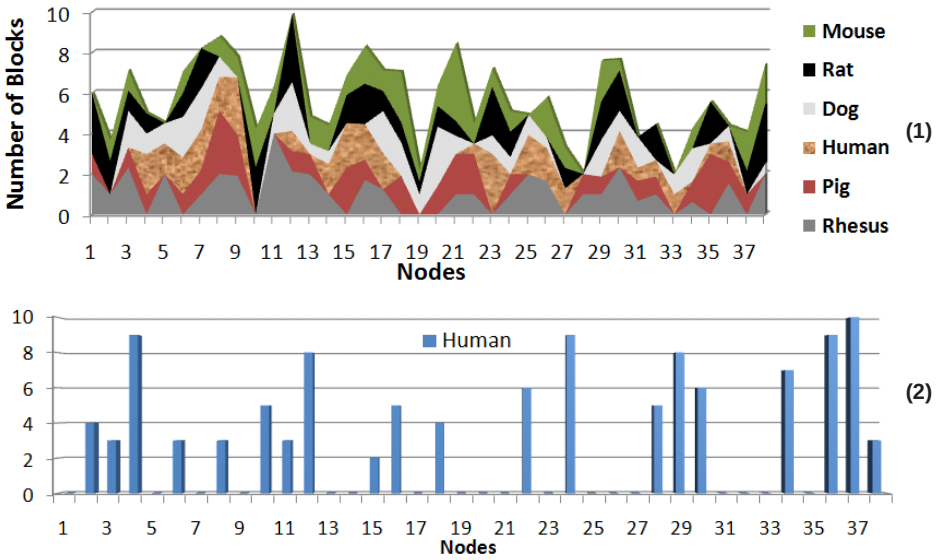


Figure 4.8: The data layout after bulk uploading six species' genome data, and the human's genome data layout.

When using the category-based uploading method, we surprisingly find that the overall data distribution is similar as what is shown in Figure 4.8. To highlight the unbalanced distribution of the related data, we quantify the degree of unbalance with $1 - \frac{\#\ of\ nodes\ having\ the\ data}{\#\ of\ nodes}$. With 20 runs using the species-based data uploading method, on average, the data of a specific species is distributed over only 53.2% nodes of the cluster. The conclusion shows that even when the data is uploaded based on the initial data grouping information, the Hadoop's random data placement is not able to achieve the maximal parallelism for the associate data.

In order to show the efficiency of the DRAW data placement strategy, Figure 4.9 plots the balanced data distribution (human) after using DRAW on our Hadoop test bed. The grouping information to generate HDAG is artificially defined as: all human data is accessed as a group. Note that we assume the human data is the single grouping data only for Figure 4.9, so that to avoid the noise from other data groups. This will be released in the following sections.

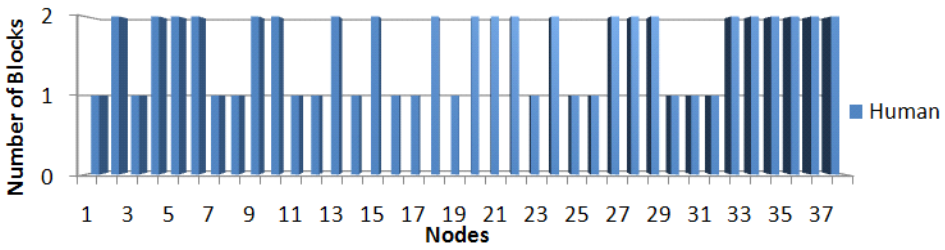


Figure 4.9: The layout of human genome data after DRAW placement.

Table 4.1: Comparison of two runs of Genome Indexing application

	Total maps	Local maps	Ratio
On DRAW	397	302	76.1%
On Random	401	189	47.1%

4.5.2 Performance Improvement of MapReduce Programs

4.5.2.1 Genome Indexing

Based on the DRAW re-organized $40GB$ genome data, we run the Bowtie indexing MapReduce program to index the human's chromosomes. Figure 4.10 shows the traces of two runs on DRAW's re-organized data and Hadoop's randomly placed data, respectively. We configure the MapReduce job according to the assumptions described in Section 4.3. The number of reducers is set as large as possible so that the reduce phase will not be the performance bottleneck. In our case, we use 39 reducers. The map phase running on DRAW's data is finished 41.7% earlier than the one running on randomly placed data, and the job's overall execution time is also improved by 36.4% when using DRAW's data. The reason is shown in Table 4.1. The MapReduce job running on the DRAW's re-organized data has 76.1% maps which benefit from having data locality, compared with 47.1% from the randomly placed data; the number of local map tasks is increased by $(320 - 189)/189 = 59.8\%$.

Note that there are still 23.9% maps which are working without having data locality even after the DATA's data re-organization. There are two reasons: first, the data grouping information the BEA algorithm used is generated from all previous MapReduce

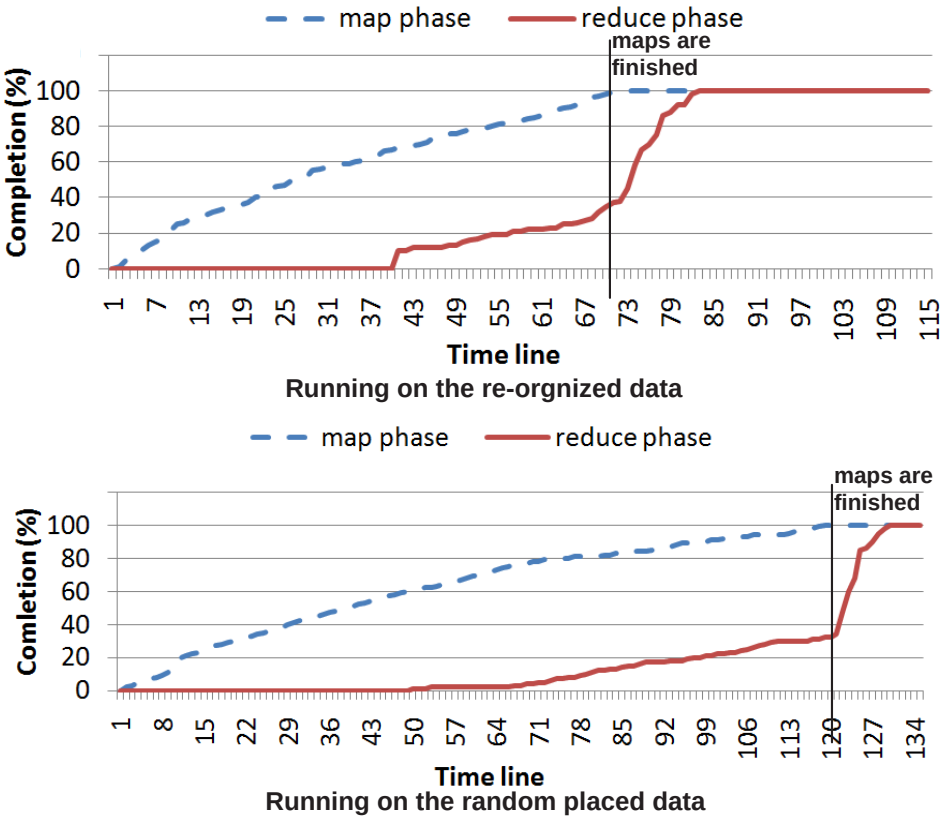


Figure 4.10: The running of Genome indexing MapReduce program on human genome data.

programs rather any specific one, and the ODPA follows ***High-Weight-First-Placed*** strategy, which means the data with higher (accumulative) grouping weights will be granted higher priority to be evenly distributed. In other words, the distribution of the non-hottest data is only optimized but may be not 100% perfect for the corresponding MapReduce programs. Second, the matrix clustering is a N-P hard problem, hence the clustered grouping matrix generated from BEA algorithm, whose time complexity is $O(n^2)$ rather than $O(n^n)$, is a pseudo-optimal solution. Adoption of BEA algorithm is a reasonable tradeoff between efficiency and accuracy. However, since the hottest data will be granted the highest priority to be clustered, the applications interested in these data can achieve the ideal parallelism. Apparently, before we run the human genome indexing application, the human data is not the hottest based on the history information; its data distribution is changed and different from that Figure 4.9 shows.

4.5.2.2 Mass Analyzer on Astrophysics Data

In the above bio-informatics applications, the data size of each species, especially for the mammals, is about $3GB$ after decompression. When using Hadoop's default $64MB$ block size, about 48 blocks are required to represent one species, which is greater than the 40 nodes in our test bed. In this section, we do experiments on smaller data sets: each particle's data is exactly $512M$, which will be split into only 8 blocks.

Our Mass Analyzer on the astrophysics data tries to calculate the average mass of each area. The results are shown in Figure 4.11. DRAW reduces the map phase by 18.2%, and the overall performance of the MapReduce program is improved by only 11.2%. It is obvious that the impact of DRAW is linearly related to the size of the required data by the MapReduce program. The less data is being accessed, the more close that random data placement can achieve maximized parallelism (which is already proved in Section 4.3). For example, given 40 nodes in the cluster and 2 maximum simultaneous map tasks on

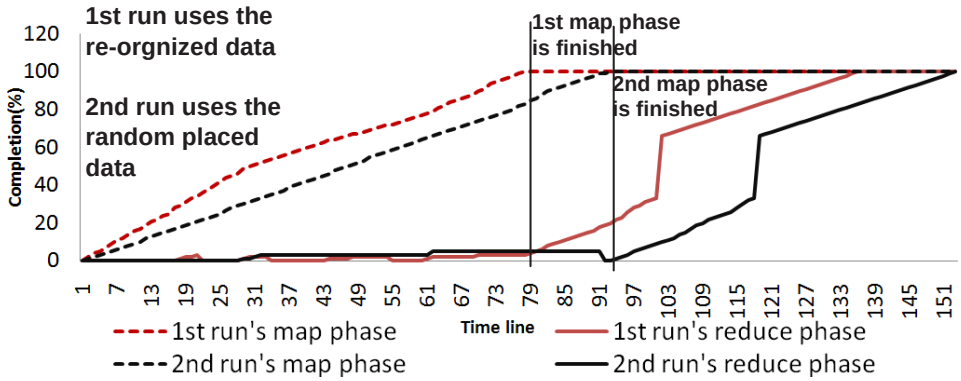


Figure 4.11: The running of Mass Analyzer on astrophysics data; the size of interested data for each run is relative small (8 blocks on average).

each node, the 8 blocks of each astrophysics data file is more likely to be balanced placed when compared to the 48 blocks of an mammal's genome data. Hence the conclusion is DRAW works better for the MapReduce programs accessing large-scale data (larger than $3GB$ for our hardware configuration).

4.5.3 Sensitivity Study: the number of replica (NR)

Table 4.2: Comparison of the experimental NHD (% of nodes holding the data) and DRAW's ideal NHD

	NR=1			**NR=2**			**NR=3**		
	Blks	E_NHD	DRAW_NHD	Blks	E_NHD	D_NHD	Blks	E_NHD	D_NHD
Stickleback	44	44.7%	100%	82	63.2%	100%	122	81.6%	100%
Opossum	48	47.4%	100%	100	73.7%	100%	150	86.8%	100%
Chicken	61	73.7%	100%	122	97.4%	100%	174	89.5%	100%
C.briggsae	13	26.3%	***34.2%***	23	42.1%	***60.5%***	34	68.4%	***89.5%***

The number of replica (NR) for each data block in Hadoop cluster is configurable. For data distribution, the more replica for each block exist, the higher possibility that the grouping data can be evenly distributed. Hence, the efficiency of DRAW on the MapReduce programs is inverse proportional to NR in the Hadoop.

In order to quantify the impact of NR on our design, we bulkily upload the $40G$ genome data to our test bed configured with $NR = 1$, $NR = 2$ and $NR = 3$, respec-

tively. Figure 4.12 shows the data distributions for four species: Stickleback, Opossum, Chicken from vertebrates, and C.briggsae from nematodes. The "% of nodes holding the data (NHD)" is directly related to the parallelism that the program accessing corresponding species can use. The results prove that, in most cases [4], NR is linearly related to the parallelism of data distribution; which means a higher degree of replica in Hadoop can mitigate the problem of unbalanced grouping-data distribution. For example, the Stickleback data is only distributed on 44.7% of the nodes in 1-replica Hadoop; when using 3-replica Hadoop, 81.5% of the nodes can provide Stickleback data.

Now we study the efficiency of DRAW for multiple replica Hadoop systems. We still use the above data. Table 4.2 shows the comparison of the experimental NHD and DRAW's ideal NHD. The NHD difference indicates the possible improvement DRAW can achieve. Note that for the three vertebrates, the number of blocks for each each species is larger than the number of nodes in our test bed, hence ideally, DRAW can distributes the grouping data on all the nodes, with 100% NHD; for the C.briggsae whose number of blocks is smaller than 40, the ideal DRAW's NHD is calculated as $\frac{\#\_of\_Blks}{\#\_of\_nodes}$, which is shown in bold font in Table 4.2. Our experimental results show that, for the 2-replica Hadoop, DRAW may improve the data distribution parallelism by 27.2% on average; for the 3-replica Hadoop, DRAW is expected to improve the parallelism by 17.6% (without considering the exception of Chicken data) on average.

4.5.4 Overhead of DRAW

The usage of the DRAW tool is similar to the Hadoop's balancer. The Hadoop administrator may manually launch the DRAW utility, and stop it at anytime; note that the data is not "optimal distributed" until the DRAW tool is finished. In this section, we

[4]There is one exception for Chicken: the data is more evenly distributed in 2-replica case than 3-replica.

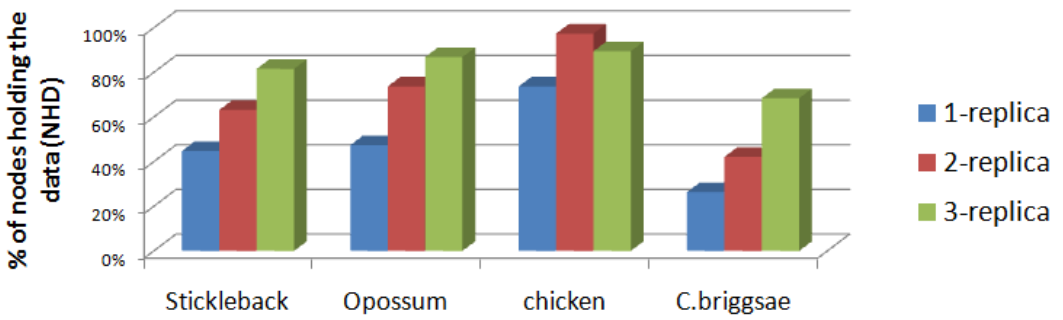


Figure 4.12: The data distributions (NHD) of four species, on 1-replica, 2-replica and 3-replica Hadoop.

quantify the overhead of running DRAW (a complete run until it is finished) on the $40GB$ genome data after 20 initial runs on our test bed cluster

The three parts of DRAW: building HDAG, building and clustering DAM, and re-organizing the data based on ODPA, have different overheads.

Building HDAG: The first step is negligible because it only scans the customized log files (only several kilobytes for 20 runs of the genome indexing program) once and records pertinent information in HDAG table.

Building and Clustering DAM: $40GB$ data will be split into about 640 blocks, based on our algorithm, the memory requirement is $6.7MB$. The BEA algorithm takes 37 seconds to cluster the 640×640 matrix.

Data Re-organization: This is the most time-consuming step in DRAW algorithm, because we have to login every *DataNode* to migrate the data/metadata/registration information. The data migration time is linearly related to the data size and the network bandwidth among the nodes. In our specific case, after 20 warm-up runs, 497 out of the 640 blocks need to be re-organized. The overall run time of the DRAW tool is $4.7min$.

The overall execution times of our genome indexing program on randomly placed data and DRAW re-organized data are $33min43sec$ and $20min37sec$, respectively. Hence the above time costs (about $5min25s$) are worthy compared to the overall performance

improvements, about 13min. This improvement can be expected for all the subsequent genome indexing programs which follow the previous access patterns.

CHAPTER 5

MAR: A NOVEL POWER MANAGEMENT FOR CMP SYSTEMS IN DATA-INTENSIVE ENVIRONMENT

In this chapter, we first launch extensive experiments which show that in a CMP system: 1) scaling down the core's frequency during its I/O wait time can provide more opportunities to save power without sacrificing performance; 2) core's waiting time for I/O operations to complete is unpredictable, unmodel-able, and depends on several factors, such as I/O type (sync or unsync), process or application level parallelism; 3) there is **no model** we could find that accurately describes the relationship between the CPU's frequency and overall performance when I/O wait time exists, because CPU frequency and I/O wait time are decoupled. As a result, power management solutions for data-intensive applications demand that: 1) considerations of each core's I/O wait status and its working and idle statuses be made; 2) accurate quantification of each status (*e.g.*, busy, idle, iowait) for accurate power-saving decisions; 3) precise description of the relationships among frequency, performance and power consumption when I/O wait factor is considered.

Then we propose an empirical rule-based power management strategy named **MAR** (modeless, adaptive, rule-based) for CMP systems. Our design can precisely **control the performance of a CMP chip to the desired set point while saving as much power as possible at run-time**. There are two primary contributions of this work.

- Comprehensive factors: while most existing control theory based works (close-loop controllers) only consider incomplete CPU statistics, MAR is designed strictly based on comprehensive experiments measuring the impacts of all the core's working

status (*e.g.* user, nice, sys, idle, iowait, irq and soft irq), and especially the I/O factor.

- Rule-based control: While most existing power saving works adopt model predictive control theories, MAR applies formal rule-based control theory [AP93] because the system (relationships among frequency, performance, and power) is too complex to be modeled when I/O wait factor is incorporated. In addition, the model-free nature of rule-based control method avoids the troublesome effort to develop accurate system models, and the risk of design errors caused by statistical inaccuracies or inappropriate approximations.

5.1 Task I: Learning the Core's Behaviors

In this section, we exploit the behaviors of each core in a CMP processor to learn the relationship among power consumption, performance, and frequency settings, as shown in Figure 5.1.

As widely shown in previous works, CPU power consumption and performance are both highly related to CPU frequency [ref04, refi]. The cubic relationship between power consumption and processor frequency, which is $CPU\ Power \propto f^3$ (f is the core frequency), is well-documented and shown in Figure 5.1.

However, the relationship between performance and frequency is difficult to be modeled: the same frequency setting may results in different response time (rt) or execution time (et) for different types of applications. The performance is related to both processor's frequency and the workload characteristics. On the other hand, the behavior of the CPU is able to illustrate the characteristics of the running workloads. More specifically, each core in a CMP has 7 working statuses [BMK02, BC05]:

- user: normal processes executing in user mode;

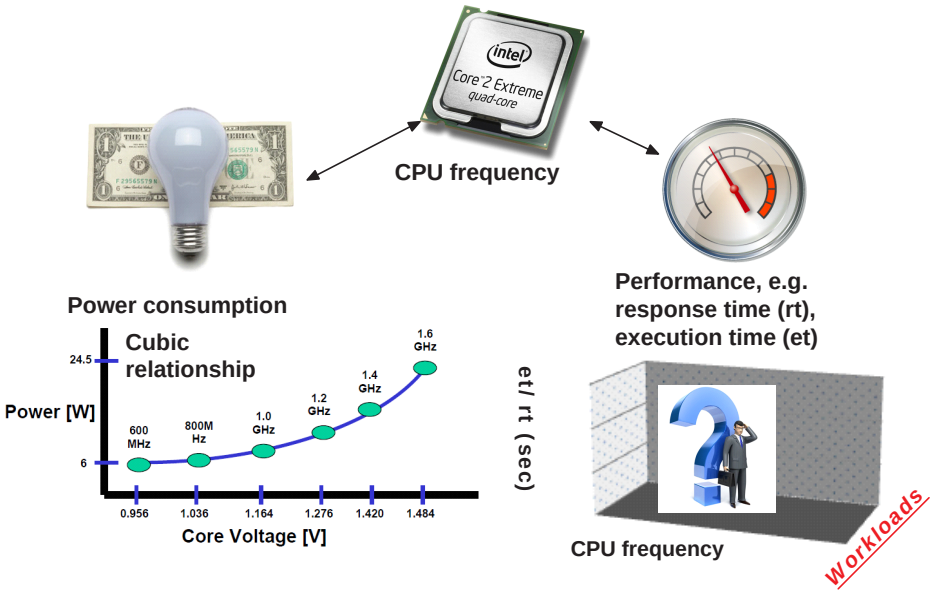


Figure 5.1: The relationship among CPU's frequency, power consumption and performance

- nice: niced processes executing in user mode;
- system: processes executing in kernel mode;
- idle: idle times;
- iowait: waiting for I/O to complete;
- irq: servicing interrupts;
- softirq: servicing soft irqs;

The durations of the core's 7 statuses completely exhibit the composition of the running workload. As a result, a relationship between performance and frequency can be denoted as Equation 5.1.

$$\begin{aligned} Executiontime &= F(frequency, workload) \\ &= F(freq., core's metrics) \\ &= F(freq., user, nice, sys, idle, iowait, irq, softirq) \end{aligned} \tag{5.1}$$

Table 5.1: L1 data cache miss, L2 cache miss and mispredictions per 1000 instructions.

	L1D miss	L2 miss	Mispredictions
gcc	14.21	3.17	5.11
mcf	130.15	36.73	15.79

We launch various applications on our test bed to learn the curve of Equation 5.1, *e.g.*, I/O bomb from Isolation Benchmark Suite (IBS) [refk], gcc and mcf benchmark from SPEC CPU 2006 suite version 1.0 [refm], TPCC running on PostgreSQL [refo]. I/O bomb uses the IOzone benchmark tool to continuously read and write to the hard disk (by writing files larger than main memory which ensures that we are not just testing memory); mcf is the most memory bound benchmark in SPEC CPU2006; gcc is cpu-intensive, as shown in Table 5.1; TPCC is a standard On-Line-Transaction-Processing (data-intensive) benchmark. The configuration details of these benchmarks can be found in Section 5.5. Our CPU for the experiment is the Quad-Core Intel Xeon E5345 2.27GHz processor, with $2 \times 4MB$ L2 cache and $1.333MHz$ FSB. The supported frequencies are $800MHz, 1.6GHz, 2.27GHz$.

5.1.1 Per-Core

Because we are using per-core level DVFS for power management, it is necessary to know the meanings of the 7 statuses for each core. We first enable only one core in the CMP processor and assign one process to run the benchmarks so that we can avoid the noise from task switches among cores. Figure 5.2 shows the overall execution time (et) of the 4 benchmarks at different frequency settings. B-I (busy-idle) model is a simple method [CSP04, FWB07] to model the relationship between et and frequency settings. In this model, $et(new) = et(old) \cdot (\frac{percent(busy) \cdot f(old)}{f(new)} + percent(idle))$, where $percent(busy)$ is the CPU busy percentage and $percent(idle)$ is the idle percentage; f and $f(new)$ are

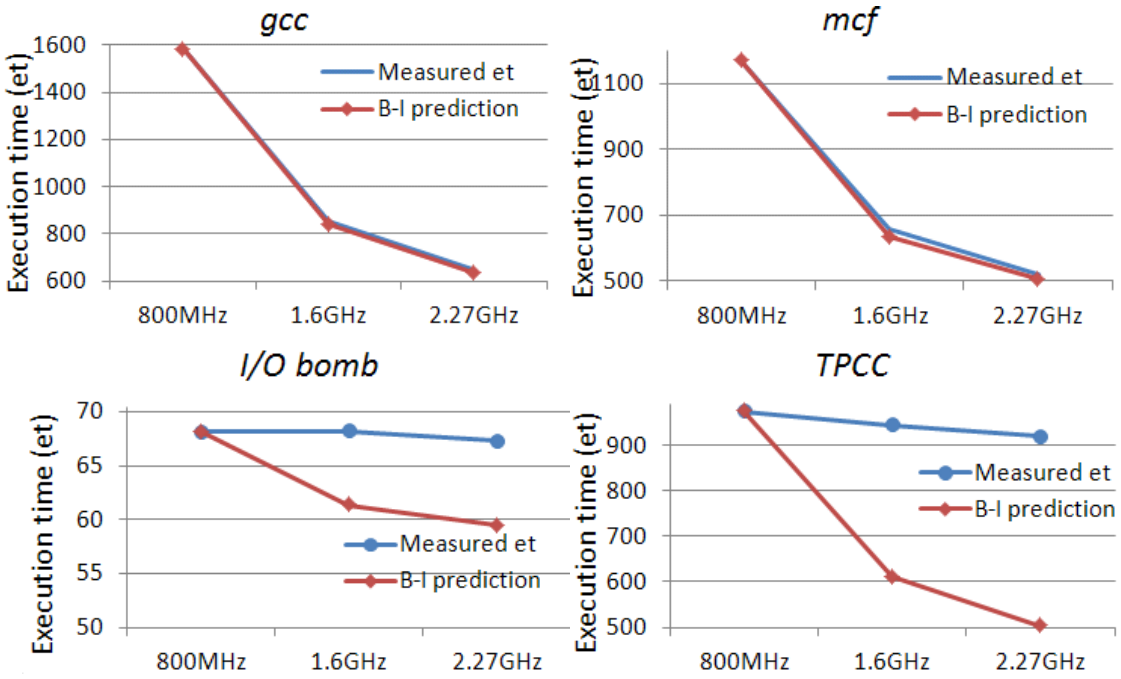


Figure 5.2: Prediction accuracy of Busy-Idle model for different workloads

the two different versions of CPU frequency settings. The predictions are based on the CMP behaviors when frequency is set as $800MHz$.

Figure 5.2 illustrates the prediction results. It is surprisingly to find that for the first two workloads, *e.g.* gcc (CPU-intensive) and mcf (memory-intensive), B-I model is accurate enough with less than 3% deviation, which is different from some previous works' [WRW05, ICM06] results. We believe this is caused by different test bed and cache-miss-penalty-reducing techniques [refe]. On the other hand, for the I/O intensive or data-intensive workloads, *e.g.* I/O bomb and TPCC, B-I model which does not consider the I/O impact will result in up to a 45% error in the model. The reason why B-I works well for CPU-intensive and memory-intensive workloads is because of the well-developed techniques to reduce cache miss penalty [refe]. However, the huge speed gap between I/O devices and processors cannot be effectively eliminated [IBC06b], which leads to the B-I model's prediction errors for I/O bomb and TPCC benchmarks.

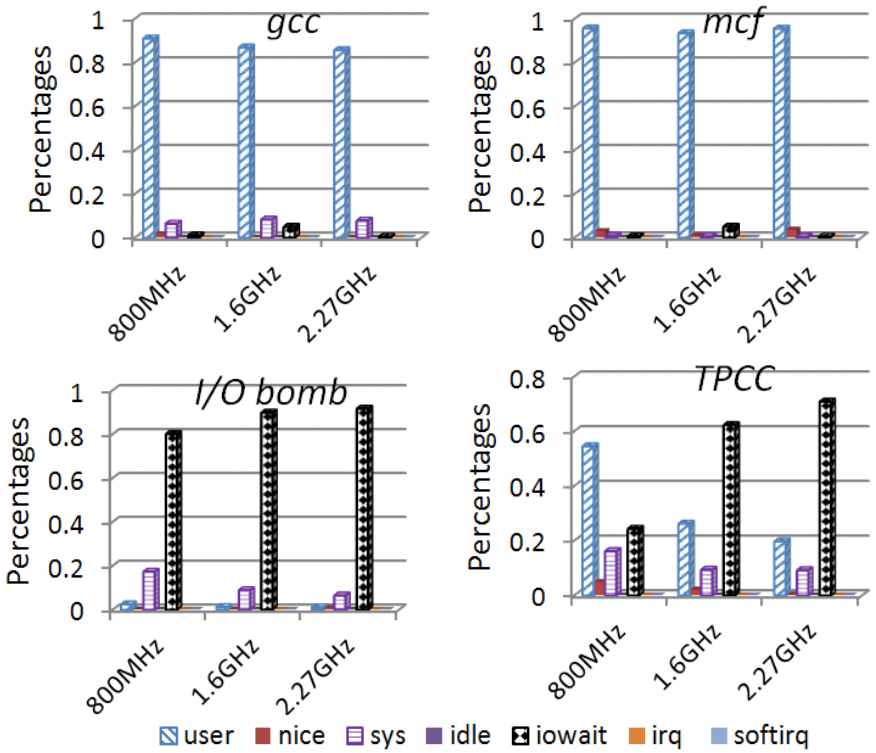


Figure 5.3: Core's statistics for different workloads

We also show the statistics of the 7 statuses during the benchmarks' running in Figure 5.3. For gcc and mcf, most of the execution time is in user mode; the cache miss and mispredictions of mcf have negligible impact on the CMP's behavior due to the built techniques reducing the cache miss penalty. For I/O bomb, I/O wait is the main latency; for data-intensive benchmark TPCC, the lower frequency will hide some of the I/O wait latency, but the latencies in both user and iowait modes can not be ignored. For all four cases, the irq and softirq latency are negligible. As a result, "user+nice+sys", "idle" and "iowait" are the three most important working statuses which could describe the CMP's behavior. So to confirm, without considering I/O wait latency, the basic B-I model may result in non-trivial modeling errors for data-intensive or I/O intensive applications.

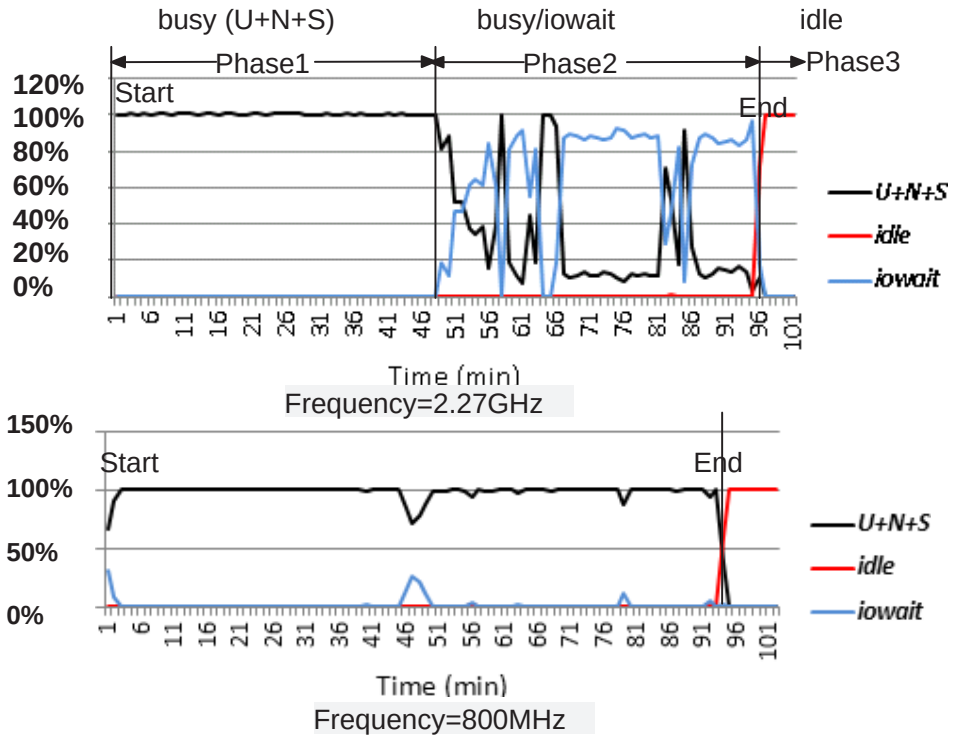


Figure 5.4: Working status trace of core0; the overall execution times are comparable for both cases.

5.1.2 Multi-Core

Due to the job scheduler in CMP processors tasks in CMP processor may be switched among the cores during its run. In order to show whether this intra-core level task switches and inter-core level job scheduling can eliminate I/O wait latency, we run 7 processes on all 4 cores in our testbed. Each process will randomly run one of the benchmarks: gcc, mcf, bzip2, gap, applu, gzip and TPCC. Each core has 3 available frequency settings: 2.27GHz, 1.6GHz and 800MHz.

We try all the possible configurations for the benchmarks [1] and record the statistics for each core. In Figure 5.4, we show the traces for core0. We omit "irq" and "softirq" based on the results of section 5.1.1, and we treat "user, nice, sys" as a group denoting the real "busy" status. When the frequency is $2.27GHz$, all the workloads are processed in

[1] core0 and core1 are on the same die so they have to be scaled together, this also applies to core2 and core3. Hence there are $3 \times 3 = 9$ different settings in total.

parallel in "phase1", the I/O wait latency could be hidden by the process-level parallelism. However in "phase2", when there are little available processes to schedule, the I/O wait latency will emerge. After all work is complete, the core will stay idle in "phase3". The traditional B-I based power management is only able to discover the chances in "phase3" and to save power by lowering the processor's voltage and frequency at the phase. However in fact, "phase2" also provides opportunities to save more power: we can lower the frequency in order to parallelize the CPU and I/O works as much as possible. As shown in the lower part of Figure 5.4, we can use "800MHz" to finish all the workloads at roughly the same time while only consuming 4.4% power when compared to the case using $2.27GHz$ frequency.

We admit that a heavy disk utilization may not necessarily result in I/O wait if there are enough parallel CPU-consuming tasks. However, the new emerging data-intensive analyses and applications lead to higher chances of having I/O latency [VBL09, KSK08, SCK06], and high I/O wait is also common in other applications [refl, refj]. As a result, I/O wait latency should be exploited in power-saving projects.

5.1.3 Analysis of I/O Wait

The iowait time is the duration of time when the processor is waiting for the I/O operation to complete, however we cannot simply consider it as the sub-category of CPU idle time. When there are only CPU idle and busy statuses, increasing the CPU frequency will linearly decrease execution time; however when taking into account the I/O wait time we have two new cases as shown in Figure 5.5. In case 1 where the CPU-consuming tasks and I/O tasks are synchronous (sequential) or blocking each other [ECC04], the I/O wait time could be treated as idle time, hence we can use the traditional B-I method, as discussed in Section 5.1.1, to model the relation between execution time and frequency.

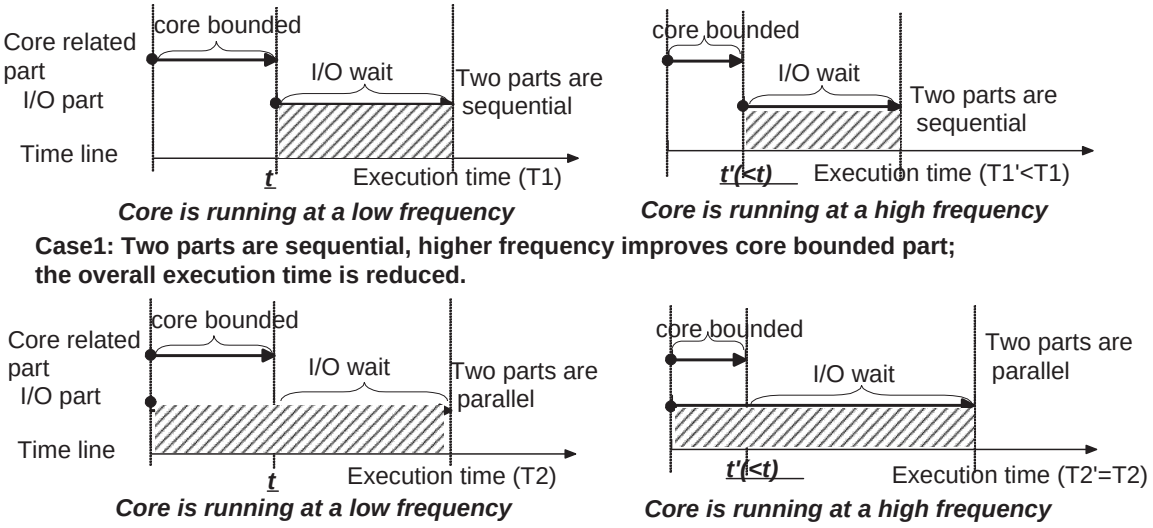


Figure 5.5: Two cases when I/O wait time exists. "Core bounded" area represents the busy status.

In case 2 where the two types of workloads are running in parallel but not well aligned, scaling CPU frequency will not affect the overall execution time [2].

Based on our comprehensive experimental results, we find that the I/O wait ratio ($\frac{iowait\ time}{overall\ time}$) could be used to distinguish the two cases with an about 5% error rate (for our specific system configuration). We introduce two "thresholds" for I/O wait ratio: th_{up} and th_{down} to quantify the Equation 5.1 as the following Equation 5.2,

$$\begin{cases} When\ scaling\ up\ freq,\ if\ I/Owait < th_{up};\ or \\ when\ scaling\ down\ freq,\ if\ I/Owait < th_{down}: \\ \quad (Case1:)\ \ rt_{new} = (\frac{1}{1+(\frac{1}{\xi}-1)P_{core}})rt_{old} \\ Otherwise:\ (Case2:)\ rt_{new} = rt_{old} \end{cases} \tag{5.2}$$

rt is response time; ξ is the ratio of the new frequency to the old one, which is $\frac{f_{new}}{f_{old}}$; P_{core} is the core's busy ratio, which is $\frac{busy\ time}{overall\ time}$. The default vale of th_{up} and th_{down} are

[2]Since we are discussing cases where I/O wait time exists, the I/O part lasts longer than the CPU-consuming part; otherwise, there will be no I/O wait time.

based on our comprehensive experimental results. Note that these two thresholds are affected by the throughput of I/O devices, L1/L2 cache hit rates, network traffic, *etc.*. A self-tuning strategy for these two thresholds are detailed in Section 5.2.3.

Equation 5.2 can be used to complete Figure 5.1. Our rule-based power management controller MAR will be designed according to the two relationships in Figure 5.1.

5.2 Task II: A Modeless, Adaptive, Rule-based (MAR) Controller Design

In this section, we introduce the design, analysis, optimizations of our rule-based power management.

5.2.1 MAR Control Model

In order to perform good power management, we have to know the relationships among CPU frequency, power consumption, and overall performance. However, as we mentioned in previous sections, after considering I/O wait factors, the relationship between the frequencies and overall performance is too complicated to be modeled (because the CPU frequency and I/O time is decoupled). Hence we adopt the formal rule-based, modeless, control system to manage the power consumption in CMP systems. The rules are derived from experimental results and could be self-tuned for different system configurations.

MAR is designed as a MIMO controller shown in Figure 5.6. Let ***SP*** denote the sampling period. RRT represents the required response times and cb represents the realtime core boundness of the workloads (core's busy ratio, $\frac{usr+nice+sys}{SP}$), rt is measured response time, w is I/O wait raito ($\frac{IOwait}{SP}$); ecb and ew are the tracking errors of core boundness ($|real\_cb - predicted\_cb|$) and I/O wait ratio ($|real\_w - predicted\_w|$), respectively; Δecb and Δew are the changing speeds of ecb and ew, respectively.

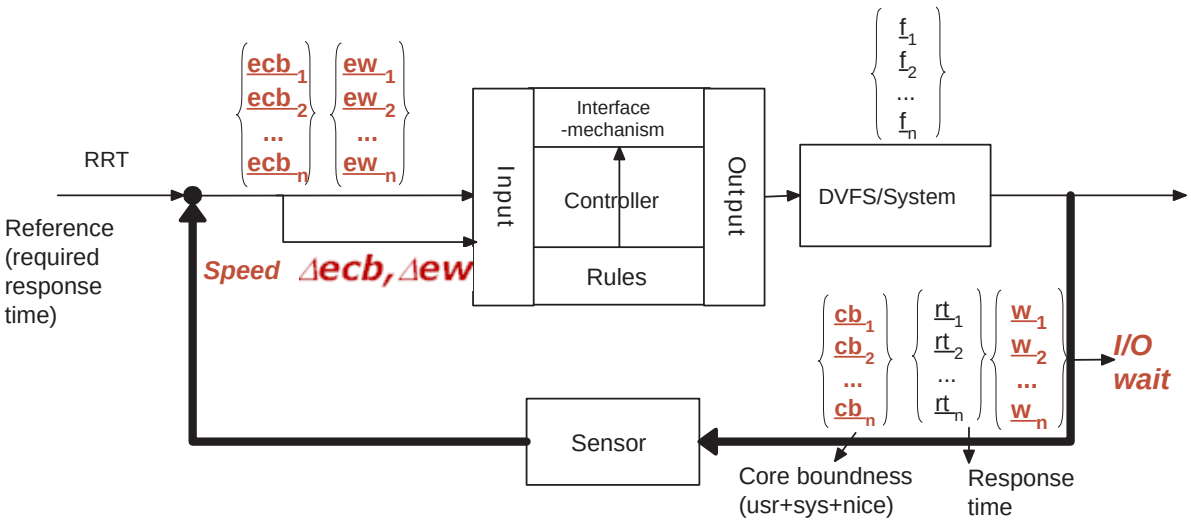


Figure 5.6: The overall architecture of MAR power management

One basic control loop is described as follows. At the end of each SP, rt, cb and w are fed back into the controller through the sensors. Based on the predicted version and the fed back version of cb and w, ecb and ew could be calculated. After that, Δecb and Δew are derived based on ecb and ew. These rt, cb, w, ecb/ew and $\Delta ecb/\Delta ew$ will be processed into the arguments P_{core}, $I/Owait$, rt_{new}, and rt_{old} of Equation 5.2. **Then MAR can determine the vector for $\xi = \frac{f_{new}}{f_{old}}$, which could be used along with the rules in Section 5.2.2 to indicate how the cores' frequencies in next SP should be updated**. Now we show how to calculate the arguments.

1) P_{core}*:* Based on cb, ecb and Δecb, MAR adopts the formal fuzzy rule-based method proposed in [BSK10] to compute the P_{core}: we first fuzzilize ecb and Δecb into linguistic values such as negative large (NL) and positive small (PS). Then we look up the knowledge base (shown in Table 5.2) to find the corresponding signal such as NM after which we convert [BSK10] the linguistic signal to a crisp value $next\_ecb$. Finally, we have $P_{core} = cb + next\_ecb$. By not only measuring the error but also tracking the change in error, MAR is **highly responsive** to status changes of P_{core}.

2) $I/Owait$**:** Based on w, ew, and Δew, $I/Owait$ is calculated in the same way of calculating P_{core}. Table 5.2 is used to derive $next\_ew$, and then $I/Owait = w + next\_ew$.

Table 5.2: Fuzzy Rule-Base to Calculate P_{core} and $I/Owait$

$e/\Delta e$	NL	NM	NS	ZE	PS	PM	PL
NL	NL	NL	NL	NL	NM	NS	ZE
NM	NL	NL	NL	NM	NS	ZE	PS
NS	NL	NL	NM	NS	ZE	PS	PM
ZE	NL	NM	NS	ZE	PS	PM	PL
PS	NM	NS	ZE	PS	PM	PL	PL
PM	NS	ZE	PS	PM	PL	PL	PL
PL	ZE	PS	PM	PL	PL	PL	PL

3) rt_{new} ***&*** rt_{old}***:*** rt_{new} should equal to the set point we want to achieve after DVFS, which is the required response time RRT. The rt_{old} represents the measured response time in current SP, so $rt_{old} = rt$.

We do not consider memory boundness in MAR. The reason is the experiments from Section 5.1.1 show that the B-I model (considering busy and idle ratio) is accurate enough for the power prediction of both CPU-intensive and memory-intensive workloads, and it is already incorporated in MAR. Here we want to focus on the more important factor: I/O wait ratio.

5.2.2 Rules

In this section, we impose a set of rules for MAR by incorporating the calculated ξ as well as the I/O wait factor. Having only ξ does not provide enough information to seize all the opportunities of saving power. These rules will guide MAR in finding the frequencies to be set in next SP. We denote $0 \leq \delta < 1$ as the user-specified performance-loss constraint.

5.2.2.1 $RTT \cdot (1 - \delta) \leq rt \leq RTT \cdot (1 + \delta)$

This is the ideal case from a performance perspective. Traditional solutions may not change the core's frequency setting, however **MAR will do a further check** whether $w > th_{down}$.

- If so, the frequency can be scaled down to a lower level to save more power without affecting rt.
- If not, scaling the frequency will result in a different rt which is deviated from RRT, so we keep using the current frequency.

5.2.2.2 $rt > RRT \cdot (1 + \delta)$

If the real response time does not meet the requirement, **MAR checks whether** $w > th_{up}$. And thus:

- If w exceeds the scaling up threshold, changing to a higher frequency will not improve the performance. Moreover, higher frequencies will result in a higher $I/Owait$, which is a waste of core resources. So as a result, MAR will keep the current frequency setting.
- If w is within the threshold, $MAF = \xi \cdot f$, where f is the current core frequency. A higher core frequency could improve rt in this case. Based on Equation 5.2, we calculate the f_{new} by using Equation 5.3.

$$f_{new} = \frac{cb}{\frac{RRT}{rt} - 1 + cb} \cdot f_{old} \tag{5.3}$$

5.2.2.3 $rt < RRT \cdot (1 - \delta)$

If the measured response time is unnecessarily better than the requirement, **there is an additional chance to scale the frequency down to save more power**. And thus:

- If $w > th_{down}$, MAR will only scale down the core frequency by one level. The reason for this "lazy" scaling is because it is difficult to know what w will be when using lower frequencies. The new w decides whether we should further scale down the frequency or not.

- If $w \leq th_{down}$, we may be able to scale down the core frequency to just meet the performance requirement while saving more power. MAR adopts aggressive scaling by using the same method shown in Equation 5.3.

We summarize the rules of MAR in Table 5.3. The detailed rules are described in Table 5.4.

Table 5.3: Rules in MAR to adjust the CPU frequency

rt vs. RRT	iowait(w)	Linguistic meaning	Lable
$RTT \cdot (1 - \delta) \leq rt \leq RTT \cdot (1 + \delta)$	$w > th_{down}$	RTT is met, I/O wait is dominating	#1
	$w \leq th_{down}$	RTT is met, changing frequency affects rt	#2
$rt > RRT \cdot (1 + \delta)$	$w > th_{up}$	RTT is not met, I/O wait is dominating	#2
	$w \leq th_{up}$	RTT is not met, scale frequency up by using cb	#3
$rt < RTT \cdot (1 - \delta)$	$w > th_{down}$	RTT is over met, I/O wait is dominating	#1
	$w \leq th_{down}$	RTT is over met, scale frequency down by using cb	#4

Table 5.4: Detailed rules description

rule #1	$f_{new} = f_{old} - 1$
rule #2	$f_{new} = f_{old}$
rule #3	$f_{new} = \frac{cb}{\frac{RRT}{rt} - 1 + cb} \cdot f_{old}$
rule #4	$f_{new} = \frac{cb}{\frac{RRT}{rt} - 1 + cb} \cdot f_{old}$

5.2.3 Self-Tuning

There are several factors affecting the thresholds th_{up} and th_{down}. For example: 1) Throughput of I/O devices. Higher I/O throughput means the same amount of data could be transferred in less "I/Owait" jiffies. 2) On-chip L1/L2 cache hit rates. The lower cache hit rate results in more memory accesses, which is much slower than cache access. Therefore, the overall processing speed of core bounded part (including both cache and memory accesses) becomes slower. 3) Noise in I/O wait time, such as network I/O traffic, file system journaling, paging/swapping, *etc.* 4) Heat and heat dissipation. When processors run too hot, they can experience errors, lock, freeze, or even burn up. It is difficult to predict the thresholds in this case, hence we adopt self-tuning methods based on the observed system behavior.

The process of self-tuning is shown in Figure 5.7.

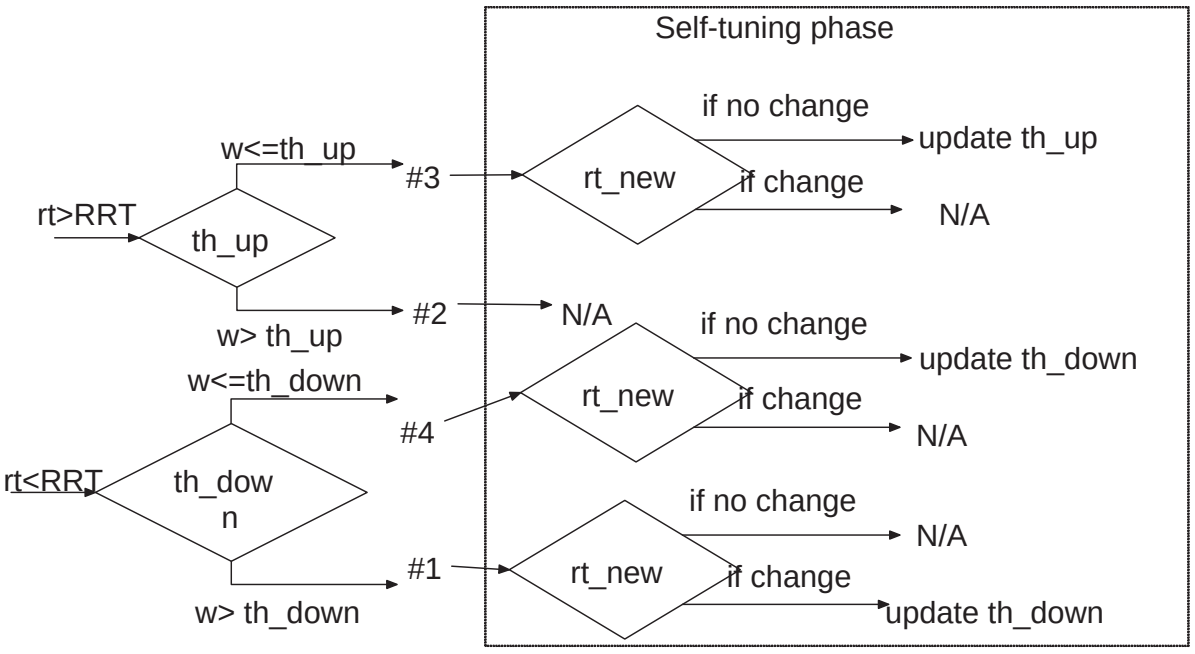


Figure 5.7: Self-tuning of I/O wait thresholds ("rt" is response time, "th" is threshold, "w" is I/O wait percentage)

When $rt > RRT$ and $w \leq th_{up}$, rule #3 is used to scale up the core frequency to meet the performance requirement. However, if the rt_{new} after the frequency scaling is same as the rt in last SP, we need to lower th_{up}.

When $rt < RRT$, rule #1 or #4 is used to scale down the core frequency to save more power. If rule #4 is applied and the rt_new does not change after the scaling, the th_{down} should be adjusted to a lower level; if rule #1 is applied and the rt_new changes, it means w should be lower than th_{down}. Hence we scale th_{down} to a higher level. In the design of MAR, we adopt "lazy" update scheme as shown in Equation 5.4 in our self-tuning method for the purpose of system stability. In Equation 5.4, "+" is used when updating the th_{up} or th_{down} to a higher level, "−" is used when updating the th_{down} to a lower level.

$$new\_th = old\_th \pm \frac{w}{2} \tag{5.4}$$

5.3 Other design issues

5.3.1 Calculating Δecb

It should be noted that the fast responsiveness of MAR may overreact to some small fluctuations, which will not result in real switches. We define them as "phantom". Therefore, a filter μ is introduced for robustness purpose: if $|ecb_i| < \mu$, the burst is phantom which should be ignored, which means the changing speed of tracking error is 0. μ is derived from our experimental experiences and could be automatically tuned during the runtime. The calculation of Δecb_i is shown in Equation 5.5.

Table 5.5: The Relationship Between Core Frequency and Performance in Six Hybrid Benchmarks

Hyb	Freqency	Exec. time	I/O wait	Core bds.	Perf. impr.	Est.Perf. Impr.	Perf. degr.	Est.Perf. Degr.
	$800MHz$	45.45	0.14	0.85	-	-	0.94	0.69
1	$1.6GHz$	44.25	0.57	0.43	1.03	1.27	1	0.88
	$2.27GHz$	44.27	0.70	0.30	1	1.10	-	-
	$800MHz$	38.95	***0.11***	0.88	-	-	0.65	0.54
2	$1.6GHz$	25.55	**0.3**	0.69	***1.54***	***1.53***	0.95	0.83
	$2.27GHz$	24.21	0.49	0.51	1.06	1.16	-	-
	$800MHz$	35.23	***0.08***	0.91	-	-	***0.49***	***0.52***
3	$1.6GHz$	17.35	***0.11***	0.89	***2.03***	***1.95***	0.95	0.78
	$2.27GHz$	16.35	0.34	0.66	1.06	1.35	-	-
	$800MHz$	28.72	***0***	1	-	-	***0.48***	***0.5***
4	$1.6GHz$	13.84	***0***	1	***2.07***	***2***	0.88	0.75
	$2.27GHz$	12.17	0.20	0.79	1.14	1.41	-	-
	$800MHz$	32.03	***0***	0.98	-	-	***0.53***	***0.54***
5	$1.6GHz$	17.27	***0.25***	0.74	***1.88***	***1.96***	0.95	0.78
	$2.27GHz$	14.76	0.53	0.47	***1.17***	***1.23***	-	
	$800MHz$	31.69	***0***	1	-	-	***0.54***	***0.52***
6	$1.6GHz$	16.59	***0.05***	0.94	***1.91***	***2***	***0.71***	***0.77***
	$2.27GHz$	12.37	***0.29***	0.71	***1.40***	***1.38***	-	-

$$\begin{cases} \Delta ecb_i = \dfrac{ecb_i}{ecb_{i-1}} & |ecb_i| \geq \mu \\ \Delta ecb_i = 0 & |ecb_i| < \mu \end{cases} \tag{5.5}$$

5.3.2 Specifying The Threshold(s)

Six hybrid benchmarks are ran on a single core in turn. The hybrid benchmarks combines CPU bombs, memory bombs and I/O bombs [refk] with different percentages. Each of them consists of integer and floating calculation, memory allocation as well as I/O operations. The results are shown in Table 5.5. For performance improvement/degradation, 1 means it is not changed with the scaled frequency; > 1 means performance is improved; < 1 means degradation.

In column "perf. impr." (performance improvement), we scale up the frequency and calculate the numbers by $\frac{old\ exec.time}{new\ exec.time}$, which is always ≥ 1. For example in Hyb1

benchmark, the performance improvement in second row is 1.03, which means the overall execution time is not improved too much when the frequency is scaled from $800MHz$ up to $1.6GHz$. Similarly, in column "perf. degr." (performance degradation), we scale down the frequency and calculate the ratio $\frac{old\ exec.time}{new\ exec.time}$, which is always ($\leq 1$).

In Table 5.5, we highlight all the cases where B-I model is applicable. For the rest cases, the noise brought by I/O wait latency eliminates the effect of scaling core frequency. Based on the list results, there is a "threshold" for frequency scaling up ("performance improvement") and another one for frequency scaling down ("performance degradation"). If the core frequency is going to a higher level: only when the I/O wait part is less than $th_{up} = 11\%$, we could use B-I model to estimate the performance with low tracking error; otherwise, the overall performance is not going to be improved. Similarly, when trying to use a lower frequency: only if the I/O wait part is less than $th_{down} = 30\%$, B-I model could be used.

5.4 Methodology

In this section, we show our experimental methodology and benchmarks, as well as the implementation details of each component in our MAR controller. ***Processor:*** We use an Quad-Core Intel Xeon E5345 2.27GHz processor, with $2 \times 4MB$ L2 cache and $1.333MHz$ FSB. The four execution cores are in two sockets. Our experiments show that core0 and core1 are in one group and core 2 and core3 are in another group. We change the DVFS levels of the 2 cores in each group together in order to have a real impact on the processor power consumption[3]. For simplicity, we treat this Quad-Core as a Dual-core, the first group as core0 and the second group as core1. Each core in

[3]Experiments show that only scaling one core's frequency in that group does not affect the overall power consumption.

the processor supports 3 DVFS levels: 800 MHz, 1.6 GHz and 2.27 GHz. The operating system is Ubuntu 9.04 (jaunty) with Linux kernel 2.6.28.

Benchmarks: We use the 3 stress tests highlighted earlier in the paper (CPU-bomb, I/O-bomb, and memory-bomb) from the Isolation Benchmark Suite (IBS) [refk], SPEC CPU 2006 suite version 1.0 [refm], and data-intensive benchmark: TPCC running on PostgreSQL [refo]. TPCC incorporates five types of transactions each with different complexity for online and deferred execution on a database system. Every transaction consists of a computing part and an I/O part. Due to the database buffer pool, updating of records will not be flushed until the pool is full. The flush results in a big write which alleviates the well known small I/O problem.

Core Statistics: Various information about kernel activity is available in the */proc/-stat* file. The first three lines in this file are the CPU's statistics, such as *user, nice, system, idle, etc.*. Since the introduction of Linux 2.6 the file also includes three additional columns: iowait, irq, and softiqr. All of these numbers together identify the amount of time the CPU has spent performing different kinds of work. Time units are in *USER_HZ* or *Jiffies*. In our $x86$ system, the default value of a jiffy is 10 ms, or 1/100 of a second.

MAR needs to collect specific core boundness information as well as I/O wait latency. Each core's boundness is the sum of the jiffies in *user, nice* and *sys* mode divided by the total number of jiffies in the last SP. Similarly, the I/O wait latency is calculated based on the iowait column.

The way to measure the real-time response time depends on the benchmarks. In the Isolation Benchmark, the response time can be monitored in I/O throughput. In TPCC, the primary metrics, transaction rate (tpmC), can be used as the response time. However, for the SPEC CPU2006 benchmarks, it is difficult to find any metrics to denote response time because there is no "throughput" concept here. Our previous experiments in Figure 5.2 show that these CPU-intensive and memory-intensive benchmarks have roughly linear relationships with core frequency. Hence we can calculate the number of

instructions which have been processed in the sampling period by multiplying the CPU time (first three fields in */proc/stat* file) by the core frequency. The result can be used as the response time metric.

DVFS Interface: We enable the Intel's SpeedStep on the BIOS and use the *cpufreq* package to implement DVFS. When using root privilege, we can echo different frequencies into the system file */sys/devices/system/cpu/cpu[X]/cpufreq/-scaling_setspeed*, where *[X]* is the index of the core. We test the overhead of scaling CPU frequencies in our platform, which is only 0.12 milliseconds on average.

Power Estimation: We use the power models from Wattch [BTM00] to estimate the processor's power consumption. Specifically, we first calculate the capacitance C and then calculate the power. After that, we consider the leakage power by multiplying $crossover\_sacling$ (1.2), and all other L1 data/inst cache and L2 cache's read miss/hit and write miss/hit power usage. Hence the power is denoted as $1.2kC \cdot frequency \cdot Voltage^2$ [ref04], where k is the co-efficient related to the computation intensity of workloads, for example, the k for computation bounded workloads is close to 1, while for I/O intensive workloads is close to 0 [SBN09]. Since we are focusing on the comparison of power consumption among MAR and other baselines, all these common parts such as 1.2, k, and C will be eliminated in the calculation. As a result, we simply use the cubic relation [ref04] to do the estimation: $Power \propto frequency^3$.

Baselines: The baselines are three previous CPU power saving works which do not incorporate I/O factors: Relax [GFF07], PID [AJ03], and GPHT [ICM06]. Relax is a simple statistical predictor and used by [GFF07] to assume the next sample behavior (core-boundness) is linearly related to previously monitored behaviors; we set the relaxation factor to 0.5 and the relax window size as 2 based on the empirical value taken from [GFF07]. PID is used in [AJ03] to predict the core's busy/idle ratio based on previously records by using Proportional Integral Derivative control algorithm; we tune $K_p = 0.4, K_i = 0.2, K_d = 0.4$ based on [AJ03]. GPHT (Global Phase History Ta-

ble based) predictor observes the historical patterns of busy/idle ratios from previously observed samples to derive the next phase's behavior. Previously learned patterns are recorded in a global table which is updated automatically. Based on [ICM06], we set the GPHT depth (history window size) to 4 and the table entries (number of patterns to be recorded) to 512. All these algorithms are implemented in RTAI3.8 [ref10] to trace the cores' behavior and predict the I/O wait in next SP.

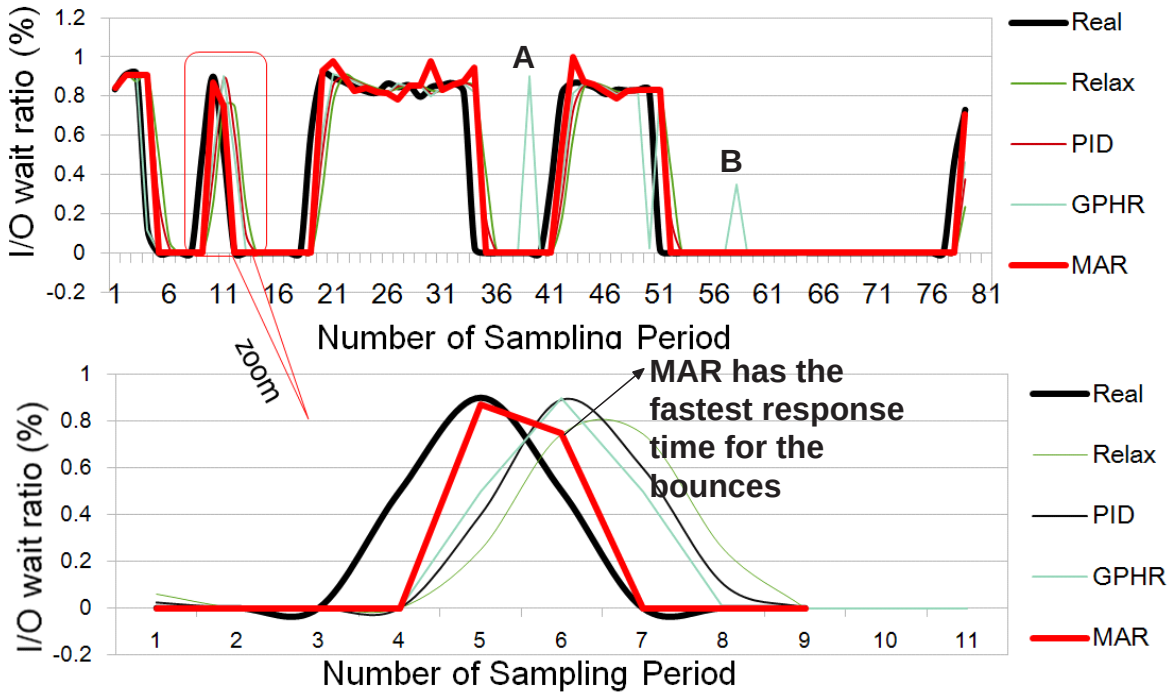
5.5 Experiments

First, we show that the model-based predictors used in the baselines are not suitable for predicting I/O wait ratio. Second, MAR is used to control the power for different types of workloads which including the CPU-intensive, memory-intensive and I/O-intensive benchmarks. The purpose is to show MAR's performance under specific environments. Third, we compare the two versions of MAR (with/without considering **I/O wait**) by running the data-intensive benchmarks, in order to highlight the impact of I/O wait in power management schemes. After that, we compare the overall efficiency of MAR and the other baselines. And in end, we briefly show the overheads of the various investigated power management schemes.

5.5.1 Modeless-ness of I/O wait

In this section, we prove that the model-based predictors such as Relax, PID, and GPHT are not suitable for I/O wait trajectory learning.

First, we implement Relax, PID, and GPHT to predict the I/O wait ratio based on their predefined models. Figure 5.8 shows their trajectories of prediction compared with MAR. We also run the test 10 times to collect the "average detecting times", as shown



Avg. time (n*sp)	Relax	PID	GPHR	MAR
Detecting the bouncing	2.5	3.5	1.85	1.25
Settling after bouncing	4	5.4	2.5	1.3
Avg. Response time	3.3	4.5	2.3	1.3

Figure 5.8: Comparison of the prediction accuracy of I/O wait ratio on a randomly picked core

in the table in Figure 5.8. It can be seen that MAR has the fastest response time (as shown in the zoomed figure in Figure 5.8). GPHT could also detect the $I/Owait$ bounce quickly is because of its aggressiveness: when there is no matched history pattern, GPHT assumes the next sample behavior is identical to the last one. But GPHT may result in some severe prediction errors like "A" and "B" spots in Figure 5.8 due to pattern changes, and GPHT has the highest overhead as shown in Section 5.5.2.4. The table in Figure 5.8 also shows that MAR achieves the shortest settling time after the deviation.

The overall response time of MAR outperforms Relax, PID, and GPHT by 2.55, 3.49, and 1.87 times, respectively.

Second, we measure the impact of SP in the prediction accuracy. Figure 5.9 shows the average prediction errors for all four algorithms when $SP = 5s$, $SP = 10s$, and $SP = 20s$. When using a smaller SP, the trajectory of the core boundness is more unstable so all the predictors have higher average prediction errors; when using larger SP, the core's behavior is more predictable due to the larger time window in each step. Slow responsive algorithms such as PID do not work well here since they are only good for the workloads with strong locality. GPHT could miss-predict because of few repetitive patterns in our experiment. In summary, MAR obtains the lowest occurrence of prediction errors for two reasons: 1) it incorporates the changing speed of the tracking error, which gives more hints for the coming trend and high responsiveness of the core's status switches; 2) it adopts the noise filter μ as shown in Equation 5.5 to reduce the unnecessary abrupt fluctuations caused by "phantom" bursty cases which is defined in our technique report.

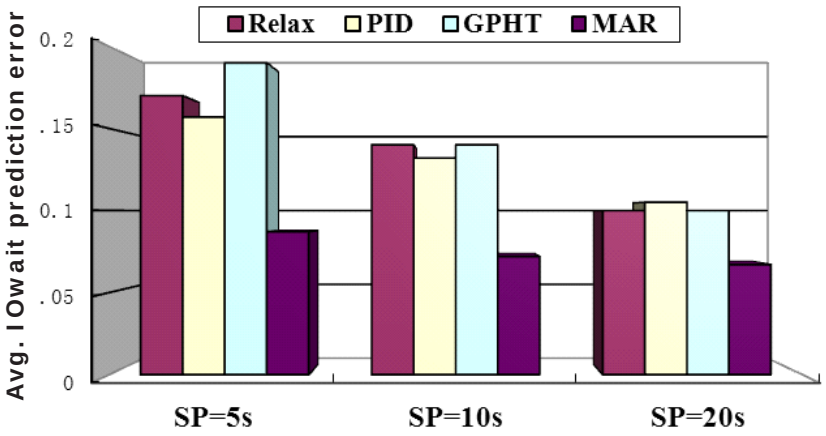


Figure 5.9: Avg. Prediction errors for different SPs

5.5.2 Power Efficiency

This set of experiments shows the power management efficiency of MAR for different types of benchmarks: gcc, mcf, bzip2, gap, applu, gzip and TPCC.

5.5.2.1 Running homogeneous workloads

In this section, we want to show MAR's power control performance when homogeneous workloads are running. For each benchmark, we use 4 threads to run 4 copies on our test bed to evaluate the MAR's performance for each specific type of workload. Here we show the results of power consumption/performance loss of MAR and the baselines: Relax, PID, GPHT and the Ideal case in Figure 5.10. In the "Ideal" case, we use the ideal DVFS settings which were calculated offline, to achieve the best power saving efficiency and the least performance loss.

Assuming the Ideal case saves the most power, MAR and the other baselines perform well when the workload has no explicit I/O operations. For gcc, mcf, bzip2, gap, applu and gzip, MAR achieves 95.4% efficiency, relative to ideal power management. The other baselines could also achieve similar control accuracy, but when running the TPCC benchmark the baselines can only achieve a $58.2 - 70.1\%$ efficiency in power saving performance, relative to the ideal case. In contrast, when considering I/O wait time as an additional opportunity to save power, MAR can still achieve 92.5% efficiency power management, relative to the ideal case. MAR outperforms the other baselines by $25.3 - 34.3\%$. The performance loss of all power management strategies is between $2\% - 3\%$. And, although, MAR has the highest performance loss of 2.8% for the TPCC benchmark, because of our aggressive power saving strategy, it is still in the safe zone [AJ03].

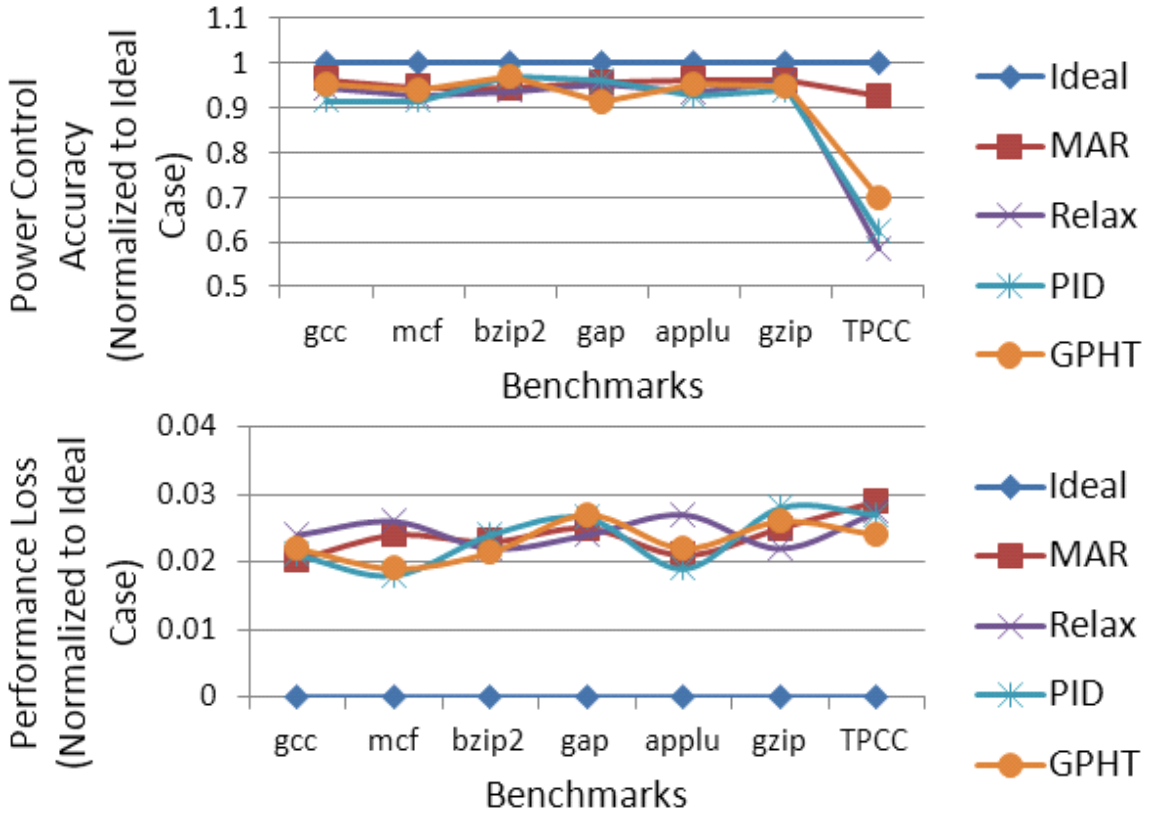


Figure 5.10: MAR's performance for various benchmarks

5.5.2.2 Running heterogeneous workloads

In this section we compare MAR with the other baselines for the case when heterogeneous workloads are running. We launch all aforementioned 7 benchmarks in parallel on our test bed. The database for the TPCC benchmark is locally set up. Figure 5.11 shows their overall DVFS results and power saving efficiency.

The upper two charts in Figure 5.11 show the frequency distribution of all management methods. Note that compared with $SP = 5s$, the trajectory of workload in $SP = 10s$ case has less fluctuations caused by the "phantom bursts". The methods lack of considering I/O factors such as Relax, PID and GPHT could not discover as many power-saving opportunities as MAR, especially in the smaller SP case.

The lower two charts in Figure 5.11 illustrate the overall power consumptions of all management methods. All the numbers are normalized to MAR which saves the most

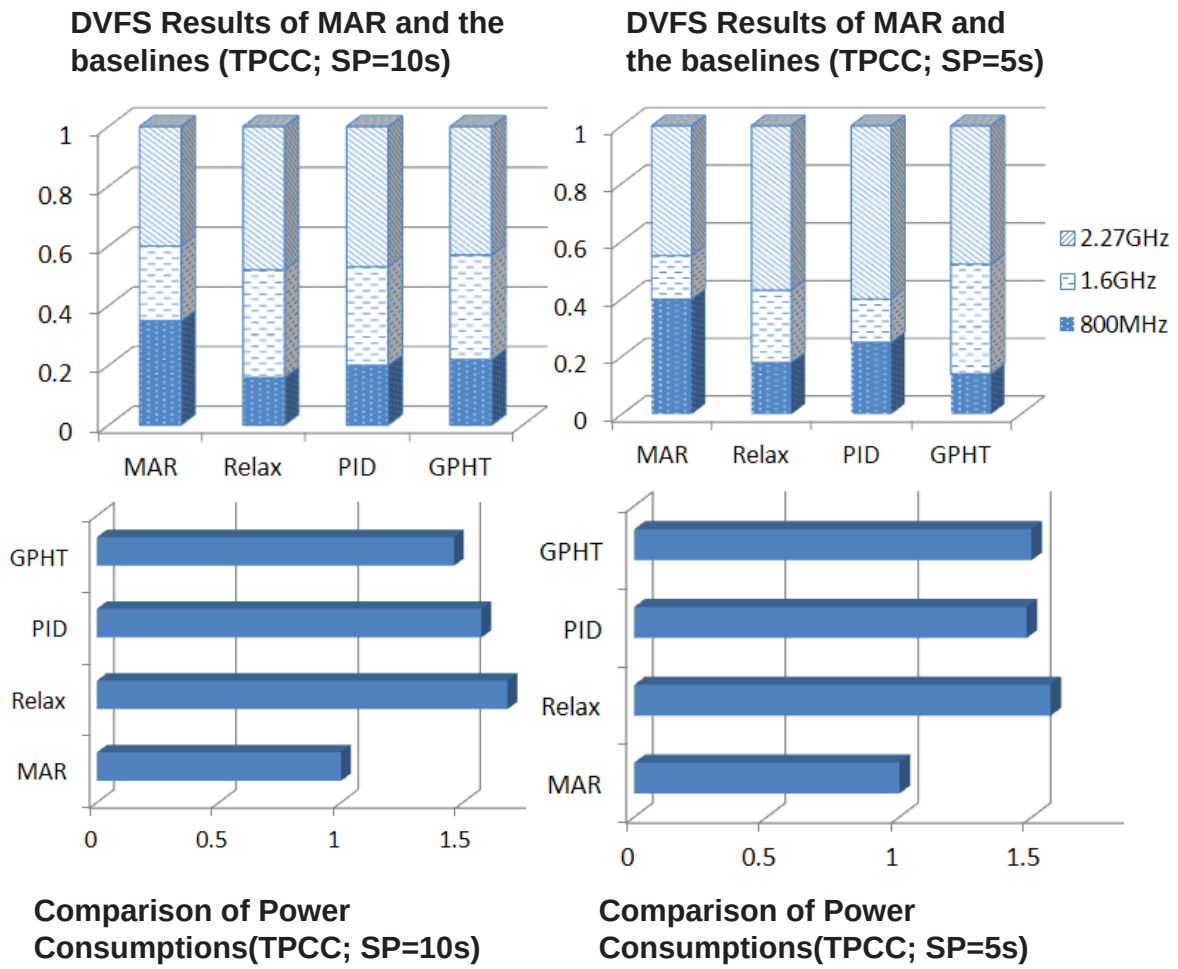


Figure 5.11: Comparison of the power management efficiency of MAR with the baselines, $SP = 10s/5s$

power. PID and GPHT perform very differently when $SP = 10s$ and $SP = 5s$. The reason is that more "phantom bursts" of the workloads (when $SP = 5s$) could affect the control accuracy. From the power saving perspective, MAR, on average ($SP = 10/5s$), saves 30.6% more than Relax, 25.9% more than PID, 21.2% more than GPHT.

5.5.2.3 The impact of I/O wait latency

In order to highlight the impact of I/O wait latency in power management, we implement an incomplete version of MAR: MAR(-W). MAR(-W) uses the same controller as as MAR

does but without considering any of the I/O factors. We use 7 threads to run gcc, mcf, bzip2, gap, applu, gzip, and TPCC in parallel. The comparison of MAR and MAR(-W) is shown in Figure 5.12.

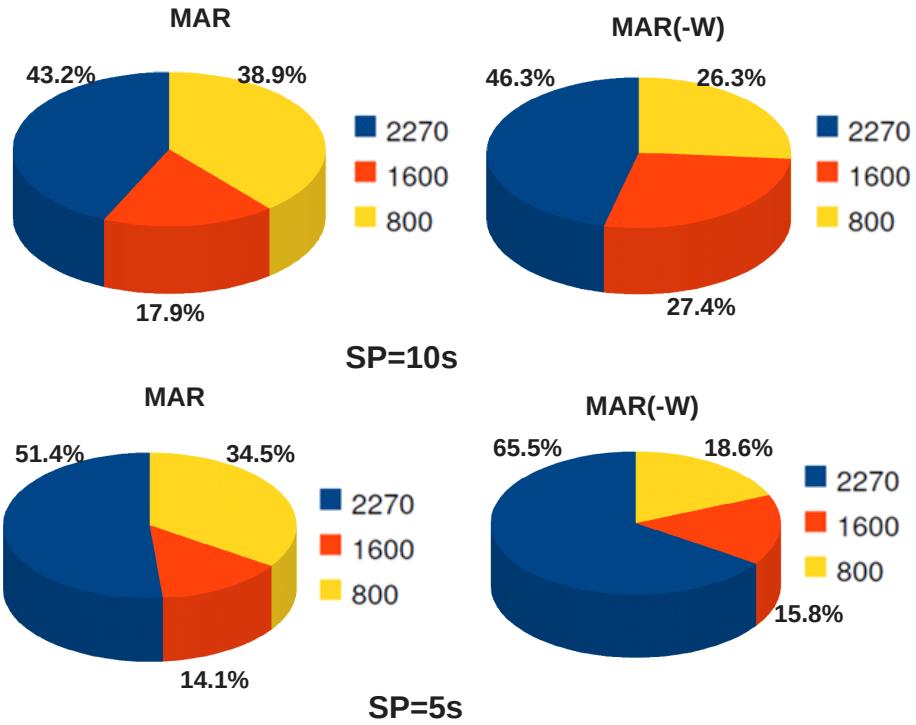


Figure 5.12: Running gcc, mcf, bzip2, gap, applu, gzip and TPCC, the DVFS results of MAR/MAR(-W), $SP = 10s/5s$

The results show that MAR is more likely to use lower frequencies than MAR(-W). The reason is that when the I/O wait exceeds the thresholds in the control period, even if the response time is close to RRT, MAR still scales down the core frequency to a lower level to save more power. Compared with MAR, MAR(-W) cannot detect the potential I/O work which is overlapped with the computing intensive work. Based on the cubic relation between frequency and power consumption, when $SP = 10s$, MAR could save 19.9% more power than MAR(-W); when $SP = 5s$, MAR saves 31.13% more power. Therefore, MAR outperforms MAR(-W) by about $20 - 30\%$.

5.5.2.4 Overhead

Table 5.6 shows the overhead of the tested methods. All of the methods are lightweight and consume less than 1% of the CPU's entire utilization for the sampling period of 10s. The GPHT controller has the highest overhead because it is indexing expensive. In contrast, MAR executes almost 9 times faster than the GPHT controller.

Table 5.6: Comparison of the overhead of different managements

	MAR	Relax	PID	GPHT
Code Size(lines)	150	50	135	600
CPU Utilization	0.11%	0.05%	0.09%	0.97%

5.5.2.5 Scalability

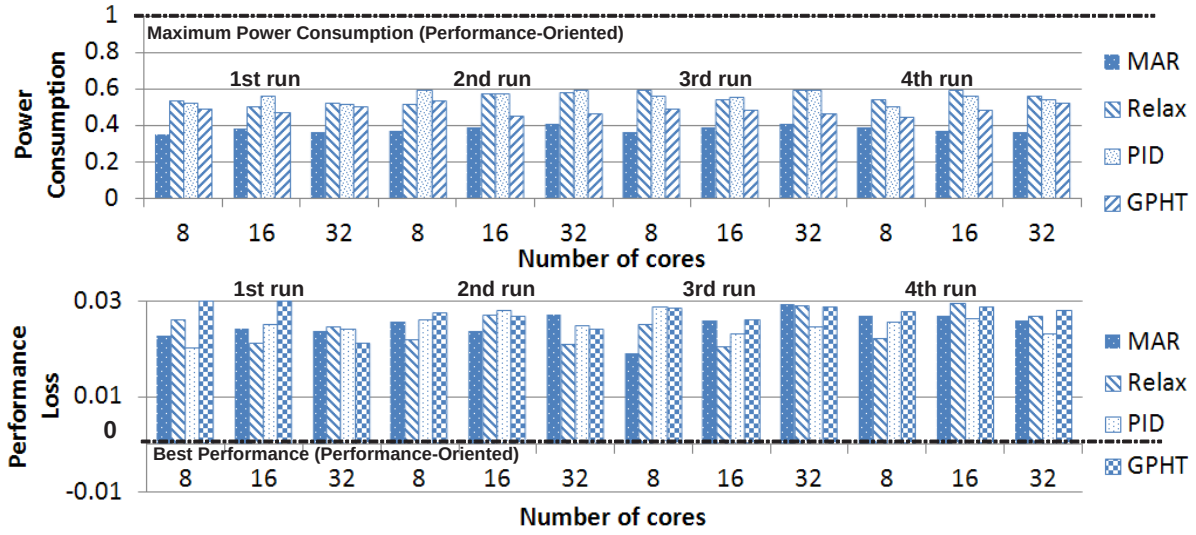


Figure 5.13: Scalability study of MAR and baselines under different number of cores in simulations

In previous subsections we have tested MAR on our testbed, which only has 4 cores and 3 available voltage-frequency settings. In order to show the scalability of MAR, we use a cycle-accurate SESC simulator [RFT05] with modifications to support per-core

level DVFS. Each core is configured as Alpha 21264 [Ec]. The processor technology is 65nm, the L1 data-cache and instruction cache are set as 64K (2 way), and 2M private L2 cache is configured. We enable wattchify and cactify [RFT05] to estimate the power change caused by DVFS scaling. In our simulation, we scale up MAR for 8, 16, 32 core processors each with private L1 and L2 hierarchy with cores placed in the middle of the die. Each core in our simulation has 3 DVFS levels (3.88GHz, 4.5GHz and 5GHz). The overhead of each DVFS scaling is set to 20μ [IBC06b]. The benchmarks we used are randomly selected SPEC 2006 benchmarks; gcc, mcf, bzip2, and the data-intensive TPCC benchmark. The number of processes are equal to the number for cores, *e.g.*, we run 2 copies of each of the 4 benchmarks when there is 8 cores. We first record the maximum power consumption and the best performance of the workloads by setting all the cores to the highest DVFS level. Then we normalize the results of MAR and other baselines to show their power management efficiency and performance losses.

Figure 5.13 plots the average power saving efficiency and the performance loss of MAR, Relax, PID, and GPHT based per-core level DVFS controllers. All the numbers are normalized to the "performance-oriented" case. With varying numbers of cores the CMP processor with MAR continually saves the most power. On average, MAR outperforms Relax, PID, and GPHT by 31.4, 32.1% and 21.3% respectively under our benchmark configuration. Also, MAR's and the other baseline's performance losses are all between $2\% - 3\%$, which confirms our observations from our test bed. Our simulation results demonstrate that MAR can precisely and stably control power while achieving good performance for CMPs with varying numbers of cores.

CHAPTER 6
RELATED WORKS

In this chapter, we will discuss the research work done in improving the transaction processing performance, and in CPU power management.

6.1 Transaction processing efficiency

TRAID deals with the performance impacts of a large log space and log latency. Tremendous amount of research has been done to improve the performance of transaction processing system by utilizing logs and storage space efficiently. In commercial database solutions like SQL Server 2008, techniques like log compression are used to mitigate the log size for mirroring databases and improve the network transfer bandwidth [RL04]. Our approach also reduces the log size, but does not incur the overhead of expensive compression algorithm.

Bulk-logged option in SQL Server reduces the penalty of logging data and metadata [SIG06]. In order to get better performance, the following operations are minimally logged and not fully recoverable: SELECT INTO, bulk-load operations, CREATE INDEX as well as text and image operations. Any-point-in-time recovery is not possible with bulk-logged option. TRAID offers partial rollback. Some other solutions include adjusting the log file size at database or application level, running hourly backups and truncating it nightly [JCM00], structuring the transaction into sub-transactions, allowing early commit of sub-transactions, and compensating transactions are provided for recovery purposes[KS03][ooH97].

To build a Database-Aware Semantically-Smart Storage (for file system [AAB06] or for database system [SBA05]), the authors investigate two techniques: First, they explore log snooping, in which the storage system observes the write-ahead log (WAL) [MHL92] records to learn the static and dynamic information; second, they explore the benefits of having the DBMS explicitly gather access statistics and write these statistics to the lower level.

Tzi-Cker Chiueh and Lan Huang [CH] mention that performance of transaction processing system is mostly determined by the amount of required physical disk I/O, which is due to database table accesses or log record writes. So they provide a high-performance transaction processing system called Charm, which aims to reduce the performance impacts of disk I/O to the minimum. This motivation is similar to our TRAID, although they focus on reducing the waiting time of conflicting transactions by making sure that all the data pages that a transaction needs be memory-resident before it is allowed to lock shared database pages.

Parity logging [SGH93] accumulates several parity updates into one bigger parity update by employing journaling techniques, so that the "write penalty" for small write can be alleviated. The XOR of the old and new block data which is being updated is logged in the fault-tolerant memory (and log disk). TRAID5 uses the parity redundancy of RAID5 in a similar way to improve the transaction processing performance.

Storage systems usually maintain redundant copies of data. Redundancy has been explored to conserve energy as in EERAID [LW04] is a novel energy-efficient RAID system architecture. With the help of redundancy, a non-blocking read of a disk can be equally transformed to a read request of another disk without hurting the overall system performance. In this way, the disk access distribution in a multi-disk system can be ultimately optimized so that near-optimal energy conservation is obtained. There are many similar works such as Diverted Accesses [PBD06], eRAID [LW06], and RIMAC [YW06], all of which have realized the necessary and potential to employ the redundancy

in the storage system. But we use the redundancy to reduce the storage workload and cost, and provide new mechanisms to make the storage system share parts of the heavy tasks which originally belong to the upper level, as a result, we can improve the storage efficiency, as well as the system throughput.

6.2 DAFA Data Management

There are several previous work which exploit the data affinity-like semantics; then organize the data in some specific ways to facilitate the future access or analysis programs. Yuan [YYL10b] proposes a data dependency-based data placement for the scientific cloud work flows, which clusters the relative data as intensively as possible hence to effectively reduce data movement during the workflow's execution. However it is different for the parallel programming frameworks, such as MapReduce. The MapReduce job performance is directly related to the data locality of each map task [refg], which in turn is related to the parallelism of the data distribution on the data nodes in the Hadoop cluster. In other words, the relative data should be distributed as evenly as possible to boost the performance of MapReduce programs.

Data diffusion [RZF08] is also designed to achieve data locality, in which the resources required for data analysis are acquired dynamically based on the demand. The required data may be acquired either "locally" or "remotely"; and cached for some time allowing more rapid responses to subsequent requires. This solution works well when similar data analysis programs are continuously launched; however when various applications with different interest localities are launched, the resources have to be dynamically distributed for each instance. In this paper, we want to design a heuristic solution based on the history information, so that the data is distributed in the way that benefits all the applications whose interest localities have been learned.

Our previous work MRAP [SMW10] is designed as a set of MapReduce APIs for the data providers who may be aware of the subsequent access patterns (affinity) of the data being uploaded. By specifying the access patterns, the data will be distributed in a corresponding way so that the best data access performance can be achieved. However in the real world, it is very difficult to know the data access patterns beforehand.

Ko [KHC10] and Yuan [YYL10a] exploit the data provenance of intermediate data in MapReduce framework, this type of data semantics can be used in two ways: 1) to provide better data fault-tolerance [KHC10]: the intermediate data may have different importance, which are quantified as the cost of reproducing them, hence they should be granted different fault-tolerant strategies; 2) to save storage capacity [YYL10a]: sometimes storing of the intermediate data is more expensive than reproducing them, therefore it is better to trade the computation cost with the storage capacity. However the way of storing or reproducing the intermediate data will not affect the applications overall performance except when a node failure happens.

6.3 Power Management

In recent years, various power management strategies have been proposed for CMP systems. From the perspective of DVFS level, previous power management schemes could be divided into two categories: chip-level and core-level power managements.

Chip-level power management uses chip-wide DVFS. In chip-level management [WRW05, ICM06, WB02, XMM05], the voltage and frequency of all cores are scaled to the same level during program execution by taking advantage of the application phase change. These techniques extensively benefit from application "phase" information that can pinpoint execution regions with different characteristics. Based on the information obtained, *Canturk, et al* [ICM06] calculate the Mem/Uop value which is quantified memory boundness

of that task. They define several CPU frequency phases in which every phase is assigned to a fixed range of $Mem/\mu op$. However, these task-oriented power management schemes do not take the advantage from per-core level DVFS.

Core-level power management means managing the power consumption of a core. [IBC06a] and [TT08] collect performance-related information by on-core performance monitoring counter (PMC) hardware. There are several limitations by using PMCs: Each CPU has a different set of available performance counters, usually with different names. Even different models in the same processor family can differ substantially in the specific performance counters available [refh]; modern superscalar processors schedule and execute multiple instructions at one time. These "in-flight"instructions can retire at any time, depending on memory access, hits in cache, stalls in the pipeline and many other factors. This can cause performance counter events to be attributed to the wrong instructions, making precise performance analysis difficult or impossible. The metrics used in MAR are simply read from the system monitoring file at run time.

Several recently proposed algorithms [IBC06a, TT08] are based on open-loop search or optimization strategies, assuming the power consumption of a CMP at different DVFS levels can be estimated accurately. This assumption may result in severe performance degradation or even power constraint violation when the workloads vary significantly from the one they used to do estimation. There are also some closed-loop solutions based on feedback control theory [CDQ05, hpc08]. The key challenge for these feedback control of power and performance is modeling, but the relationships among frequency, performance, and power consumption are too complex to be accurately modeled when running I/O-intensive workloads.

Some works [IBC06b, WRW05, ICM06] focus on non-I/O-intensive workloads so they consider the memory-boundness as the scaling referece. Their solution ignored the CPU's I/O wait time which should not be ignored in data-intensive environment. Other works [GFF07, CSP04] incorporate CPU's I/O factor into their power management solutions.

They divide every workload into "on-chip" and "off-chip" parts, in which the later one is irrelevant to CPU's frequency, and I/O wait time is categorized into the "off-chip" part along with the idle time. However, without considering the application level parallelism, they simply quantify the CPU's I/O wait latency as the application's required I/O time. This cannot be applied when the I/O time of one application is hidden by parallelizing other CPU-consuming applications or other "on-chip" processes.

Some recently proposed power managements use model predictive controllers, such as MPC, PID control models. These works are working on different levels (cluster, large scale data center, CMP), such as *DEUCON* [WJL07], [hpc08] and [WMW09]. They make an assumption that the actual execution times of real-time tasks are equal to their estimated execution times, and their online-predictive model will cause significant error in spiky cases due to slow-settling from deviation. Moreover, their control architecture allows degraded performance since they do not include the performance metrics into the feedback. [LWK05] tries to satisfy QoS-critical systems but their assumption is maintaining the same CPU utilization guarantees the same performance. It is not true for the CPU unrelated works, such as the data-intensive or I/O-intensive workloads.

Rule-based control theory [Wan97] is widely used in machine control [SCY09, TL09], and it has the advantage that the solution to the problem can be cast in terms that human operators can understand, so that their experience can be used in the design of the controller. It also reduces the development time/cycle, simplifies design complexity as well as implementation, and improves control performance [refq].

CHAPTER 7
CONCLUSION AND FUTURE WORK

In the following subsections, we describe the contributions of our work and the future work.

7.1 Contributions of Transactional RAID

In Chapter 3, we presented the results of our Transactional RAID (TRAID) work on top of BerkelyDB and PostgreSQL. TRAID is designed for transaction processing applications. It exploits the existing information redundancies in RAID and database systems to minimize the log size. We have implemented TRAID5 and TRAID10 systems for erasure coded disk array and replica based disk array, respectively.

- Transactional RAID has bridged the gap between Database systems and the underlying RAID storage systems. By considering the data redundancy at both levels (**temporal redundancy** at Database level, and **spatial redundancy** at RAID level), TRAID eliminates the overlapped information to reduce the log latency and accelerate the transaction commit. We proved Database+TRAID and Database+RAID can obtain the same level of data reliability and data availability, as well as the ACID requirement. At the same time, Our extensive results demonstrate that for throughput, TRAID outperforms RAID by $43.24-69.5\%$ for various workloads [SSW11].

- We implemented TRAID5 and TRAID10 by modifying the corresponding RAID code in Linux kernel version 2.6.11 on a Dell Precision 690 (Intel Xeon E5345 - 2.33GHz/4.0GB RAM). 5 uniform 250G SATA disk drives with 7200 rpm rotation speed are installed. We created soft (T)RAID5 (with 4 disks – 3 data disks and 1 parity disk, another disk was used as log disk) and (T)RAID10 (with 4 disks, the last disk is the log disk).

- In order to have a fair evaluation of TRAID, we used three benchmarks: a commercial benchmark for transaction processing evaluation: TPC-C [170], and two modified versions of TPC-C as micro benchmarks. TPCC simulates an Online Transaction Processing (OLTP) database environment and is the standard benchmark to evaluate transaction processing performance. Based on the implementation of standard TPC-C, we developed a special version of TPC-C for our test, named BTPC-C1 (Biased TPC-C benchmark1). In BTPC-C1, the key values in the queries and updates were changed from a uniformly random distribution to a biased distribution in the form of 90/10 rules. In this way, we increase the access locality so that the resulting workload is more sensitive to lock content delay, and the log-lock content delay. The third benchmark aims to test the performance of TRAID with a write-intensive workload, called BTPC-C2 (Biased TPC-C benchmark2). In BTPC-C2, we shield all the read-only transactions in TPC-C. Because read requests in TRAID and RAID are identical, read intensive transactions may obviate the performance improvement. Therefore, by using BTPC-C2, we can explore the advantages of TRAID for the transactions with dominant update requests.

- We also considered the locking-level issue: it is known that there are page-level locking and record-level locking in database. We want to show the performance improvement of TRAID is not limited to any specific locking level. Hence the block size of a TRAID-parity is set to 512 Bytes, which is same as the default page size in Berkeley DB and PostgreSQL. In our experiments, the page size is 512

Bytes, the record size in WAREHOUSE Table is about 480 Bytes, in CUSTOMER Table is over 700 Bytes, and in STOCK Table is about 420 Bytes; while in other 5 tables, the record sizes range from 100 Bytes to 200 Bytes. 92% of the transactions will read/write the first three tables. For the majority of time, only 1 record can be fit in 1 page, which means the page level locking in our experimental configuration is comparable to record level locking.

item Our results demonstrated that TRAID performs similarly for both page-level logging and record-level logging: for throughput, TRAID outperforms RAID by $43.24 - 69.5\%$ for various workloads; it also saves on log space by $28.57 - 35.48\%$, and outperforms RAID by about 20% in throughput when "Group Commit" is enabled. Finally, we show that TRAID outperforms RAID from 28.7% to 35.7% during the recovery.

7.2 Future Work

We have designed TRAID5 and TRAID10, in future we would like to extend this work to the more complicated RAID levels. For example, double-parity RAID or parity-based RAID6— such as RDP [CEG04]—maintains two parities P and $P^{'}$. P is same as the RAID5 parity and $P^{'}$ is used for the recovery of second disk failure. The spatial recovery requirements are different for RAID5 and RAID6, but the temporal recovery (undo/redo on a particular drive at time domain) provided by TRAID-parity Q is the same. Hence, only P parity is used to calculate the TRAID-parity Q. Hence the feasibility of porting TRAID idea to other RAID levels is proved.

7.3 Contribution of DRAW

The default random data placement in a MapReduce/Hadoop cluster does not take into account data grouping semantics. This could cluster many grouped data into a small number of nodes, which limits the data parallelism degree and results in performance bottleneck. In order to solve the problem, a new data-grouping-aware data placement (DRAW) scheme is developed. DRAW captures runtime data grouping patterns and distributes the grouped data as evenly as possible. There are three phases in DRAW: learning data grouping information from system logs, clustering the data-grouping matrix, and re-organizing the grouping data. We also theoretically prove that the inefficiency of Hadoop's random placement method. Our experimental results show that for two representative MapReduce applications – Genome Indexing and Astrophysics, DRAW can significantly improve the throughput of local map task execution by up to 59.8%, and reduce the execution time of map phase by up to 41.7%. The overall MapReduce job response time is reduced by 36.4%.

7.4 Future Work

As we analyzed in our design, DRAW currently is designed for the single-replica per rack Hadoop systems. Although we experimented the performance of DRAW in multi-replica per rack Hadoop configurations, it is still necessary to theoretically quantify how the number of replica per rack affects DRAW's performance. Moreover, we need to find more real applications having interest locality. In the current version, we do not quantify how often or how many interest locality exists in real world applications.

7.5 Contributions of MAR

In Chapter 5, we proposed an empirical rule-based power management strategy named **MAR** (modeless, adaptive, rule-based) for Chip Multi-processors. Our approach reduces the processors' power consumption while maintains the required performance.

- The observations from our extensive experiments provide a better understanding of the relationship among the frequency, performance, and power consumption in a CMP processor: 1) scaling down the core's frequency during its I/O wait time can provide more opportunities to save power without sacrificing performance; 2) core's waiting time for I/O operations to complete is unpredictable, unmodel-able, and depends on several factors, such as I/O type (sync or unsync), instruction or process level parallelism; 3) there is **no model** we could find that accurately describes the relationship between the CPU's frequency and overall performance when I/O wait time exists, because CPU frequency and I/O wait time are decoupled. As a result, power management solutions for data-intensive applications demand that: 1) considerations of each core's I/O wait status and its' working and idle statuses be made; 2) accurate quantification of each status (*e.g.*, busy, idle, iowait) for accurate power-saving decisions; 3) precise description of the relationship among frequency, performance and power consumption when I/O wait factor is considered be made.

- Comprehensive factors: while most existing control theory based works (close-loop controllers) only consider incomplete CPU statistics, MAR is designed strictly based on comprehensive experiments measuring the impacts of all the core's working status (*e.g.* user, nice, sys, idle, iowait, irq and soft irq), and especially the I/O factor.

- Rule-based control: While most existing power saving works adopt model predictive control theories, MAR applies formal rule-based control theory [AP93] because the

system (relationships among frequency, performance, and power) is too complex to be modeled when I/O wait factor is incorporated. In addition, the model-free nature of rule-based control method avoids the troublesome effort to develop accurate system models, and the risk of design errors caused by statistical inaccuracies or inappropriate approximations.

- Our MAR is a modeless, adaptive, rule-based power management scheme in multi-core systems to manage the power consumption while maintain the required performance. "Modeless" reduces the complexity of system modeling as well as the risk of design errors caused by statistical inaccuracies or inappropriate approximations. "Adaptive" allows MAR to adjust the control methods based on the real-time system behaviors. The rules in MAR are derived from experimental observations and operators' experience, which provide a more accurate and practical way to describe the system behaviors. "Rule-based" architecture also reduces the design development cycle and control overhead, simplifies design complexity. MAR controller is highly responsive (including short detective time and settling time) to the workload bouncing by incorporating more comprehensive control references (*e.g.*, changing speed, I/O wait). Noise filters are used in MAR to reduce the unnecessary abrupt fluctuations caused by "phantom" bursty cases. Empirical results on a physical testbed show that our control solution can provide precise power control, as well as high power efficiency for optimized system performance compared to four existing solutions. Based on our comprehensive experiments, MAR could outperform the baseline methods by $22.5 - 32.5\%$ in power saving efficiency, and maintains comparable performance loss about $1.8\% - 2.9\%$.

7.6 Future Work

- We would like to apply other modern pattern recognition technologies, such as Support Vector Machine (SVM), Model Predictive Control (MPC) to the CMP power management work. Because different tools have different focuses, they may be better for different workloads or system configurations.

- We would also like to apply our modeless, rule-based idea into memory power management, which has been proved as the most power-consuming part in computer systems [Men06].

- We also plan to design a joint power management strategy for CPU, memory, or even disks. There was no previous work demonstrating how each of these three parts affects others or how they affect the overall power consumption together.

LIST OF REFERENCES

[170] "TPC benchmark C standard specification, revision 5." www.tpc.org/tpcc/spec/tpcc_current.pdf.

[AAB06] A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, L. N. Bairavasundaram, T. E. Denehy, F. I. Popovici, V. Prabhakaran, and M. Sivathanu. "Semantically-smart disk systems: past, present, and future." *SIGMETRICS Perform. Eval. Rev.*, pp. 29–35, 2006.

[AAV08] S. R. Alam, P. K. Agarwal, and J. S. Vetter. "Performance characteristics of biomolecular simulations on high-end systems with multi-core processors." *Parallel Comput.*, **34**(11):640–651, 2008.

[AJ03] M. S. S. R. C. L. S. A. Varma, B. Ganesh and B. Jacob. "A control-theoretic approach to dynamic voltage scheduling." In *Proc. International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES 2003)*, San Jose CA, 2003.

[AP93] P. J. Antsaklis and K. M. Passino, editors. *An introduction to intelligent and autonomous control*. Kluwer Academic Publishers, Norwell, MA, USA, 1993.

[BC05] D. Bovet and M. Cesati. *Understanding The Linux Kernel*. Oreilly & Associates Inc, 2005.

[BDB] "Berkeley DB Documentation." www.oracle.com/technology/documentation/berkeley-db/db/api_c/env_set_flags.html.

[BMK02] M. Beck, R. Magnus, and U. Kunitz. *Linux Kernel Internals with Cdrom*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[BR06] A. Bhatkar and J. L. Rana. "Estimating neutral divergence amongst Mammals for Comparative Genomics with Mammalian scope." In *Proceedings of the 9th International Conference on Information Technology*, pp. 3–6, Washington, DC, USA, 2006. IEEE Computer Society.

[BS04] W. Bartlett and L. Spainhower. "Commercial fault tolerance: a tale of two systems." *Dependable and Secure Computing, IEEE Transactions on*, **1**(1):87–96, Jan.-March 2004.

[BSK10] C. Basaran, M. H. Suzer, K.-D. Kang, and X. Liu. "Robust fuzzy CPU utilization control for dynamic workloads." *J. Syst. Softw.*, **83**(7):1192–1204, 2010.

[BTM00] D. Brooks, V. Tiwari, and M. Martonosi. "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations." In *In Proceedings of the 27th Annual International Symposium on Computer Architecture*, pp. 83–94, 2000.

[Bur05] D. K. Burleson. *Oracle Tuning: The Definitive Reference*. Rampant TechPress, North Carolina, USA, 2005.

[CDQ05] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam. "Managing server energy and operational costs in hosting centers." In *SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pp. 303–314, New York, NY, USA, 2005. ACM.

[CEG04] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar. "Awarded Best Paper! – Row-Diagonal Parity for Double Disk Failure Correction." In *FAST '04: Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, pp. 1–14, Berkeley, CA, USA, 2004. USENIX Association.

[CGS09] A. M. Caulfield, L. M. Grupp, and S. Swanson. "Gordon: using flash memory to build fast, power-efficient clusters for data-intensive applications." In *ASPLOS '09: Proceeding of the 14th international conference on Architectural support for programming languages and operating systems*, pp. 217–228, New York, NY, USA, 2009. ACM.

[CH] T.-C. Chiueh and L. Huang. "Configuring the IBM Enterprise Storage Server for Oracle OLTP Applications,." citeseer.ist.psu.edu/197890.html.

[Che09] S. Chen. "FlashLogging: exploiting flash devices for synchronous logging performance." In *SIGMOD '09: Proceedings of the 35th SIGMOD international conference on Management of data*, pp. 73–86, New York, NY, USA, 2009. ACM.

[CM02] A. T. Chamillard and L. D. Merkle. "Evolution of an introductory computer science course: the long haul." *J. Comput. Small Coll.*, **18**:144–153, October 2002.

[CSP04] K. Choi, R. Soma, and M. Pedram. "Fine-Grained Dynamic Voltage and Frequency Scaling for Precise Energy and Performance Trade-Off Based on the Ratio of Off-Chip Access to On-Chip Computation Times." In *DATE '04: Proceedings of the conference on Design, automation and test in Europe*, p. 10004, Washington, DC, USA, 2004. IEEE Computer Society.

[DG08] J. Dean and S. Ghemawat. "MapReduce: simplified data processing on large clusters." *Commun. ACM*, **51**:107–113, January 2008.

[DKO84] D. J. DeWitt, R. H. Katz, F. Olken, L. D. Shapiro, M. R. Stonebraker, and D. Wood. "Implementation techniques for main memory database systems." In *SIGMOD '84: Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pp. 1–8, New York, NY, USA, 1984. ACM.

[DPJ03] C. E. Dyreson, T. B. Pedersen, and C. S. Jensen. "Incomplete information in multidimensional databases." pp. 282–309, 2003.

[Dum04] A. Dumitriu. "X and Y (number 5)." In *ACM SIGGRAPH 2004 Art gallery*, SIGGRAPH '04, pp. 28–, New York, NY, USA, 2004. ACM.

[Ec] O. N. Ec-Rjrza-Te. "Alpha 21264 Microprocessor Hardware Reference Manual.".

[ECC04] K. Elmeleegy, A. Chanda, A. L. Cox, and W. Zwaenepoel. "Lazy asynchronous I/O for event-driven servers." In *ATEC '04: Proceedings of the annual conference on USENIX Annual Technical Conference*, pp. 21–21, Berkeley, CA, USA, 2004. USENIX Association.

[FAA08a] A. Flores, J. L. Aragón, and M. E. Acacio. "An energy consumption characterization of on-chip interconnection networks for tiled CMP architectures." *J. Supercomput.*, **45**:341–364, September 2008.

[FAA08b] A. Flores, J. L. Aragón, and M. E. Acacio. "An energy consumption characterization of on-chip interconnection networks for tiled CMP architectures." *J. Supercomput.*, **45**(3):341–364, 2008.

[FM07] J. Fischer and R. Majumdar. "Ensuring consistency in long running transactions." In *ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pp. 54–63, New York, NY, USA, 2007. ACM.

[FSS07] A. Fedorova, M. Seltzer, and M. D. Smith. "Improving Performance Isolation on Chip Multiprocessors via an Operating System Scheduler." In *PACT '07: Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques*, pp. 25–38, Washington, DC, USA, 2007. IEEE Computer Society.

[FWB07] X. Fan, W. dietrich Weber, and L. A. Barroso. "Power Provisioning for a Warehouse-sized Computer." In *In Proceedings of ISCA*, 2007.

[GFF07] R. Ge, X. Feng, W.-c. Feng, and K. W. Cameron. "CPU MISER: A Performance-Directed, Run-Time System for Power-Aware Clusters." In *ICPP '07: Proceedings of the 2007 International Conference on Parallel Processing*, p. 18, Washington, DC, USA, 2007. IEEE Computer Society.

[GR92] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.

[GR93] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.

[GSK03] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. "Reducing Disk Power Consumption in Servers with DRPM." *Computer*, **36**:59–66, December 2003.

[GZ99] N. Gorla and K. Zhang. "Deriving Program Physical Structures Using Bond Energy Algorithm." In *Proceedings of the Sixth Asia Pacific Software Engineering Conference*, APSEC '99, pp. 359–, Washington, DC, USA, 1999. IEEE Computer Society.

[HG09] R. S. Holmes and E. Goldberg. "Brief communication: Computational analyses of mammalian lactate dehydrogenases: Human, mouse, opossum and platypus LDHs." *Comput. Biol. Chem.*, **33**:379–385, October 2009.

[HL05] Y. Hahn and B. Lee. "Identification of nine human-specific frameshift mutations by comparative analysis of the human and the chimpanzee genome sequences." *Bioinformatics*, **21**:186–194, January 2005.

[HP07] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach, 4th Edition*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[hpc08] *Cluster-level feedback power control for performance optimization*. IEEE Computer Society, 2008.

[HRV09] V. Hanumaiah, R. Rao, S. Vrudhula, and K. S. Chatha. "Throughput optimal task allocation under thermal constraints for multi-core processors." In *DAC '09: Proceedings of the 46th Annual Design Automation Conference*, pp. 776–781, New York, NY, USA, 2009. ACM.

[IBC06a] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi. "An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget." In *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 347–358, Washington, DC, USA, 2006. IEEE Computer Society.

[IBC06b] C. Isci, A. Buyuktosunoglu, C. yong Cher, P. Bose, and M. Martonosi. "An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget." In *in Proc. Intl Symp. Microarch. (MICRO*, pp. 347–358, 2006.

[ICM06] C. Isci, G. Contreras, and M. Martonosi. "Live, Runtime Phase Monitoring and Prediction on Real Systems with Application to Dynamic Power Management." In *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 359–370, Washington, DC, USA, 2006. IEEE Computer Society.

[JCM00] S. Jones, S. J. Cunningham, R. J. McNab, and S. J. Boddie. "A transaction log analysis of a digital library." *Int. j. on Digital Libraries*, **3**(2):152–169, 2000.

[Kar09] S. Karayi. "Server Energy and Efficiency Report 2009." 2009.

[KBG08] A. Krioukov, L. N. Bairavasundaram, G. R. Goodson, K. Srinivasan, R. Thelen, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dussea. "Parity lost and parity regained." In *FAST'08: Proceedings of the 6th USENIX Conference on File and Storage Technologies*, pp. 1–15, Berkeley, CA, USA, 2008. USENIX Association.

[KHC10] S. Y. Ko, I. Hoque, B. Cho, and I. Gupta. "Making cloud intermediate data fault-tolerant." In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pp. 181–192, New York, NY, USA, 2010. ACM.

[KS03] R. Karlsen and T. Strandenæs. "Trigger-Based Compensation in Web Service Environments." In *ICEIS (1)*, pp. 487–490, 2003.

[KSK08] M. Kandemir, S. W. Son, and M. Karakoy. "Improving I/O performance of applications through compiler-directed code restructuring." In *FAST'08: Proceedings of the 6th USENIX Conference on File and Storage Technologies*, pp. 1–16, Berkeley, CA, USA, 2008. USENIX Association.

[LGC06] J. G. Liu, M. Ghanem, V. Curcin, C. Haselwimmer, Y. Guo, G. Morgan, and K. Mish. "Achievements and Experiences from a Grid-Based Earthquake

Analysis and Modelling Study." In *Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, E-SCIENCE '06, pp. 35–, Washington, DC, USA, 2006. IEEE Computer Society.

[lla04] "TPCC-UVa: A free, open-source implementation of the TPC-C Benchmark." 2004.

[LM07] S.-W. Lee and B. Moon. "Design of flash-based DBMS: an in-page logging approach." In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pp. 55–66, New York, NY, USA, 2007. ACM.

[LMP08] S.-W. Lee, B. Moon, C. Park, J.-M. Kim, and S.-W. Kim. "A case for flash memory ssd in enterprise database applications." In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 1075–1086, New York, NY, USA, 2008. ACM.

[LW04] D. Li and J. Wang. "EERAID: energy efficient redundant and inexpensive disk array." In *EW11: Proceedings of the 11th workshop on ACM SIGOPS European workshop*, p. 29, New York, NY, USA, 2004. ACM.

[LW06] D. Li and J. Wang. "eRAID: A Queueing Model Based Energy Saving Policy." In *MASCOTS '06: Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation*, pp. 77–86, Washington, DC, USA, 2006. IEEE Computer Society.

[LWK05] C. Lu, X. Wang, and X. Koutsoukos. "Feedback Utilization Control in Distributed Real-Time Systems with End-to-End Tasks." *IEEE Trans. Parallel Distrib. Syst.*, **16**(6):550–561, 2005.

[Men06] A. Mendelson. "Memory management challenges in the power-aware computing era." In *Proceedings of the 5th international symposium on Memory management*, ISMM '06, pp. 1–2, New York, NY, USA, 2006. ACM.

[MH94] C. Mohan and D. Haderle. "Algorithms for flexible space management in transaction systems supporting fine-granularity locking." In *EDBT '94: Proceedings of the 4th international conference on extending database technology*, pp. 131–144, New York, NY, USA, 1994. Springer-Verlag New York, Inc.

[MHL92] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz. "ARIES: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging." *ACM Trans. Database Syst.*, **17**(1):94–162, 1992.

[MN06] S. L. Min and E. H. Nam. "Current trends in flash memory technology: invited paper." In *ASP-DAC '06: Proceedings of the 2006 Asia and South Pacific Design Automation Conference*, pp. 332–333, Piscataway, NJ, USA, 2006. IEEE Press.

[MZ06] E. Malinowski and E. Zimányi. "A conceptual solution for representing time in data warehouse dimensions." In *APCCM '06: Proceedings of the 3rd Asia-Pacific conference on Conceptual modelling*, pp. 45–54, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[OH07] C. O'Hanlon. "A Conversation with John Hennessy and David Patterson." *Queue*, **4**(10):14–22, 2007.

[ooH97] O. G. ovlen, O. T. ornsen, and S.-O. Hvasshovd. "Compensation-Based Query Processing in On-Line Transaction Processing Systems." In *BNCOD 15: Proceedings of the 15th British National Conferenc on Databases*, pp. 38–53, London, UK, 1997. Springer-Verlag.

[OV99] M. T. Özsu and P. Valduriez. *Principles of distributed database systems (2nd ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1999.

[PBA07] Z. N. J. Peterson, R. Burns, G. Ateniese, and S. Bono. "Design and implementation of verifiable audit trails for a versioning file system." In *FAST '07: Proceedings of the 5th USENIX conference on File and Storage Technologies*, pp. 20–20, Berkeley, CA, USA, 2007. USENIX Association.

[PBD06] E. Pinheiro, R. Bianchini, and C. Dubnicki. "Exploiting redundancy to conserve energy in storage systems." In *SIGMETRICS '06/Performance '06: Proceedings of the joint international conference on Measurement and modeling of computer systems*, pp. 15–26, New York, NY, USA, 2006. ACM.

[PP01] A. N. Packer and S. M. Press. *Configuring and Tuning Databases on the Solaris Platform*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[PY08] L. Peng and Q. Yongfu. "Impacts of Land Surface and Sea Surface Temperatures on the Onset Date of South China Sea Summer Monsoon." In *Proceedings of the 2008 International Workshop on Education Technology and Training & 2008 International Workshop on Geoscience and Remote Sensing - Volume 01*, pp. 277–280, Washington, DC, USA, 2008. IEEE Computer Society.

[refa] "Bowtie: An ultrafast memory-efficient short read aligner.".

[refb] "Burrows-Wheeler transform.".

[refc] "The Cosmic Data ArXiv.".

[refd] "DFS should place one replica per rack.".

[refe] "Five Ways to Reduce Miss Penalty.".

[reff] "The Hadoop Distributed File System: Architecture and Design.".

[refg] "Hadoop Tutorial Introduction.".

[refh] "Hardware Performance Counter Basics.".

[refi] "Introducing the 45nm Next-Generation Intel Core Microarchitecture.".

[refj] "Is I/O Wait a Measure of CPU Utilization or Idle Time?".

[refk] "Isolation Benchmark Suite.".

[refl] "Processor utilization and wait I/O – a look inside.".

[refm] "SPEC CPU2006.".

[refn] "The Terasort Benchmark.".

[refo] "TPCC-UVa: A free, open-source implementation of the TPC-C Benchmark.".

[refp] "UCSC Genome Bioinformatics Site.".

[refq] "Why Use Fuzzy Logic?".

[ref04] "Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor." 2004.

[ref06] "AMD Cool'n'Quiet Technology." 2006.

[ref09] "IBM EnergyScale for POWER6 Processor-Based Systems." 2009.

[ref10] "RTAI - the RealTime Application Interface for Linux." 2010.

[RFT05] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, S. Sarangi, P. Sack, K. Strauss, and P. Montesinos. "SESC simulator.", January 2005. http://sesc.sourceforge.net.

[RL04] B. Rácz and A. Lukács. "High Density Compression of Log Files." In *DCC '04: Proceedings of the Conference on Data Compression*, p. 557, Washington, DC, USA, 2004. IEEE Computer Society.

[RSG10] M. Rodriguez-Martinez, J. Seguel, and M. Greer. "Open Source Cloud Computing Tools: A Case Study with a Weather Application." In *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*, CLOUD '10, pp. 443–449, Washington, DC, USA, 2010. IEEE Computer Society.

[RZF08] I. Raicu, Y. Zhao, I. T. Foster, and A. Szalay. "Accelerating large-scale data exploration through data diffusion." In *Proceedings of the 2008 international workshop on Data-aware distributed computing*, DADC '08, pp. 9–18, New York, NY, USA, 2008. ACM.

[SB03] D. Shasha and P. Bonnet. *Database tuning: principles, experiments, and troubleshooting techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[SBA05] M. Sivathanu, L. N. Bairavasundaram, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. "Database-aware semantically-smart storage." In *FAST'05: Proceedings of the 4th conference on USENIX Conference on File and Storage Technologies*, pp. 18–18, Berkeley, CA, USA, 2005. USENIX Association.

[SBN09] H. Solar, R. Berenguer, J. de No, n. Gurutzeaga, I U. Alvarado, and J. Legarda. "A fully integrated 23.2dBm P1dB CMOS power amplifier for the IEEE 802.11a with 29 *Integr. VLSI J.*, **42**(1):77–82, 2009.

[Sch09] M. C. Schatz. "CloudBurst." *Bioinformatics*, **25**:1363–1369, June 2009.

[SCK06] S. W. Son, G. Chen, M. Kandemir, and F. Li. "Energy savings through embedded processing on disk system." In *ASP-DAC '06: Proceedings of the 2006 Asia and South Pacific Design Automation Conference*, pp. 128–133, Piscataway, NJ, USA, 2006. IEEE Press.

[SCY09] T. Shaocheng, L. Changying, and L. Yongming. "Fuzzy adaptive observer backstepping control for MIMO nonlinear systems." *Fuzzy Sets Syst.*, **160**(19):2755–2775, 2009.

[SG07] B. Schroeder and G. A. Gibson. "Understanding disk failure rates: What does an MTTF of 1,000,000 hours mean to you?" *Trans. Storage*, **3**(3):8, 2007.

[SGH93] D. Stodolsky, G. Gibson, and M. Holland. "Parity logging overcoming the small write problem in redundant disk arrays." In *ISCA '93: Proceedings of the 20th annual international symposium on Computer architecture*, pp. 64–75, New York, NY, USA, 1993. ACM.

[SHS01] C. A. Stein, J. H. Howard, and M. I. Seltzer. "Unifying File System Protection." In *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, pp. 79–90, Berkeley, CA, USA, 2001. USENIX Association.

[SIG06] R. Sears, C. van Ingen, and J. Gray. "To BLOB or Not To BLOB: Large Object Storage in a Database or a Filesystem?" Technical Report MSR-TR-2006-45, University of California at Berkeley, Microsoft, April, June 2006.

[SLZ07] M. Specht, R. Lebrun, and C. P. E. Zollikofer. "Visualizing shape transformation between chimpanzee and human braincases." *Vis. Comput.*, **23**:743–751, August 2007.

[SMW10] S. Sehrish, G. Mackey, J. Wang, and J. Bent. "MRAP: a novel MapReduce-based framework to support HPC analytics applications with access patterns." In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pp. 107–118, New York, NY, USA, 2010. ACM.

[SSW11] P. Shang, S. Serish, and J. Wang. "TRAID: Exploiting Temporal Redundancy and Spatial Redundancy to Boost Transaction Processing Systems Performance." *IEEE Transactions on Computers*, **99**(PrePrints), 2011.

[TG09] S. Tripathi and R. S. Govindaraju. "Change detection in rainfall and temperature patterns over India." In *Proceedings of the Third International Workshop on Knowledge Discovery from Sensor Data*, SensorKDD '09, pp. 133–141, New York, NY, USA, 2009. ACM.

[Tho05] A. Thomasian. "Reconstruct versus read-modify writes in RAID." *Inf. Process. Lett.*, **93**(4):163–168, 2005.

[TL09] S. Tong and Y. Li. "Observer-based fuzzy adaptive control for strict-feedback nonlinear systems." *Fuzzy Sets Syst.*, **160**(12):1749–1764, 2009.

[TPC] "HP Integrity rx6600 achieves highest 4p/8c Microsoft Windows/SQL 2005 TPC-C performance result." http://h71028.www7.hp.com/ERC/downloads/c00760409.pdf.

[TT08] R. Teodorescu and J. Torrellas. "Variation-Aware Application Scheduling and Power Management for Chip Multiprocessors." In *ISCA '08: Proceedings of the 35th International Symposium on Computer Architecture*, pp. 363–374, Washington, DC, USA, 2008. IEEE Computer Society.

[VBL09] V. Vishwanath, R. Burns, J. Leigh, and M. Seablom. "Accelerating tropical cyclone analysis using LambdaRAM, a distributed data cache over wide-area ultra-fast networks." *Future Gener. Comput. Syst.*, **25**(2):184–191, 2009.

[Wan97] L.-X. Wang. *A course in fuzzy systems and control.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1997.

[WB02] A. Weissel and F. Bellosa. "Process cruise control: event-driven clock scaling for dynamic power management." In *CASES '02: Proceedings of the 2002 international conference on Compilers, architecture, and synthesis for embedded systems*, pp. 238–246, New York, NY, USA, 2002. ACM.

[WJL07] X. Wang, D. Jia, C. Lu, and X. Koutsoukos. "DEUCON: Decentralized End-to-End Utilization Control for Distributed Real-Time Systems." *IEEE Trans. Parallel Distrib. Syst.*, **18**(7):996–1009, 2007.

[WLO01] R. K. Wong, F. Lam, and M. A. Orgun. "Modelling and Manipulating Multidimensional Data in Semistructured Databases." *World Wide Web*, **4**(1-2):79–99, 2001.

[WMW09] Y. Wang, K. Ma, and X. Wang. "Temperature-constrained power control for chip multiprocessors with online model estimation." In *ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture*, pp. 314–324, New York, NY, USA, 2009. ACM.

[WRW05] Q. Wu, V. J. Reddi, Y. Wu, J. Lee, D. Connors, D. Brooks, M. Martonosi, and D. W. Clark. "A Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance." In *In MICRO 38: Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture*, pp. 271–282. IEEE Computer Society, 2005.

[XMM05] F. Xie, M. Martonosi, and S. Malik. "Bounds on power savings using runtime dynamic voltage scaling: an exact algorithm and a linear-time heuristic approximation." In *ISLPED '05: Proceedings of the 2005 international symposium on Low power electronics and design*, pp. 287–292, New York, NY, USA, 2005. ACM.

[Yad07] H. Yadava. *The Berkeley DB Book*. Apress, Berkely, CA, USA, 2007.

[YG05] M. T. Yourst and K. Ghose. "Incremental Commit Groups for Non-Atomic Trace Processing." In *MICRO 38: Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture*, pp. 67–80, Washington, DC, USA, 2005. IEEE Computer Society.

[YW06] X. Yao and J. Wang. "RIMAC: a novel redundancy-based hierarchical cache architecture for energy efficient, high performance storage systems." In *EuroSys '06: Proceedings of the ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, pp. 249–262, New York, NY, USA, 2006. ACM.

[YXR06] Q. Yang, W. Xiao, and J. Ren. "TRAP-Array: A Disk Array Architecture Providing Timely Recovery to Any Point-in-time." *SIGARCH Comput. Archit. News*, **34**(2):289–301, 2006.

[YY01] K. H. Yeung and T. S. Yum. "Dynamic Multiple Parity (DMP) Disk Array for Serial Transaction Processing." *IEEE Trans. Comput.*, **50**(9):949–959, 2001.

[YYL10a] D. Yuan, Y. Yang, X. Liu, and J. Chen. "A cost-effective strategy for intermediate data storage in scientific cloud workflow systems." pp. 1–12, May 2010.

[YYL10b] D. Yuan, Y. Yang, X. Liu, and J. Chen. "A data placement strategy in scientific cloud workflows." *Future Gener. Comput. Syst.*, **26**:1200–1214, October 2010.

[ZYN07] L. Q. Zhou, Z. G. Yu, P. R. Nie, F. F. Liao, V. V. Anh, and Y. J. Chen. "Log-correlation Distance And Fourier Transform With Kullback-Leibler Divergence Distance For Construction Of Vertebrate Phylogeny Using Complete Mitochondrial Genomes." In *Proceedings of the Third International*

Conference on Natural Computation - Volume 02, ICNC '07, pp. 304–308, Washington, DC, USA, 2007. IEEE Computer Society.

[ZZL09] B. Zhang, N. Zhang, H. Li, F. Liu, and K. Miao. "An Efficient Cloud Computing-Based Architecture for Freight System Application in China Railway." In *Proceedings of the 1st International Conference on Cloud Computing*, CloudCom '09, pp. 359–368, Berlin, Heidelberg, 2009. Springer-Verlag.