applied sciences

MDPI

Article

Concerto: Dynamic Processor Scaling for Distributed Data Systems with Replication

Jinsu Lee and Eunji Lee *

School of Artificial Intelligence Convergence, Soongsil University, 369 Sangdoro, Dongjak-gu, Seoul 06978, Korea; wjswls735@naver.com
* Correspondence: ejlee@ssu.ac.kr; Tel.: +82-02-820-0952

Abstract: A surge of interest in data-intensive computing has led to a drastic increase in the demand for data centers. Given this growing popularity, data centers are becoming a primary contributor to the increased consumption of energy worldwide. To mitigate this problem, this paper revisits DVFS (Dynamic Voltage Frequency Scaling), a well-known technique to reduce the energy usage of processors, from the viewpoint of distributed systems. Distributed data systems typically adopt a replication facility to provide high availability and short latency. In this type of architecture, the replicas are maintained in an asynchronous manner, while the master synchronously operates via user requests. Based on this relaxation constraint of replica, we present a novel DVFS technique called `Concerto`, which intentionally scales down the frequency of processors operating for the replicas. This mechanism can achieve considerable energy savings without an increase in the user-perceived latency. We implemented `Concerto` on Redis 6.0.1, a commercial-level distributed key-value store, demonstrating that all associated performance issues were resolved. To prevent a delay in read queries assigned to the replicas, we offload the independent part of the read operation to the fast-running thread. We also empirically demonstrate that the decreased performance of the replica does not cause an increase of the replication lag because the inherent load unbalance between the master and replica hides the increased latency of the replica. Performance evaluations with micro and real-world benchmarks show that Redis saves 32% on average and up to 51% of energy with `Concerto` under various workloads, with minor performance losses in the replicas. Despite numerous studies of the energy saving in data centers, to the best of our best knowledge, `Concerto` is the first approach that considers clock-speed scaling at the aggregate level, exploiting heterogeneous performance constraints across data nodes.

Keywords: distributed systems; database systems; energy efficiency; power management

Citation: Lee, J.; Lee, E. Concerto: Dynamic Processor Scaling for Distributed Data Systems with Replication. *Appl. Sci.* **2021**, *11*, 5731. https://doi.org/10.3390/app11125731

Academic Editor: Jason K. Levy

Received: 1 June 2021
Accepted: 17 June 2021
Published: 21 June 2021

1. Introduction

Data centers play a pivotal role in this age of big data. As more applications depend on data-intensive technologies, such as artificial intelligence, bioinformatics, and data analytics, the demand for data centers continues to increase. Despite this increase in popularity, the energy use per computation in data centers has decreased significantly over the past decade. This promising shift stems from the energy efficiency progress that has been made during this time. Virtualization technologies have led to a large reduction in energy use by reducing the number of servers operating in data centers [1], and advanced power management schemes for processors have contributed to less energy per computation [2]. Nevertheless, energy efficiency continues to be important for data centers; in the U.S., the electricity used by data centers accounted for 1.3% of worldwide energy production in 2010, and these centers generate more than 43 million tons of CO_2 emissions per year, equal to 2% of the worldwide figure. The data centers are also estimated to have consumed approximately 1% of worldwide energy use in 2020 [2]. Given that data centers are energy-intensive and that the demand for their services continues to grow rapidly, additional optimization techniques are required at least to maintain the current flat increase trend in energy use [3–5].

In line with above arguments, this paper takes advantage of an opportunity to reduce the energy used by data centers by perspicaciously applying the DVFS (Dynamic Voltage and Frequency Scaling) technique in distributed data systems. DVFS scales down the frequency and voltage of the processors when the performance constraints are not stringent, instilling a trade-off the performance and savings. State-of-the-art distributed data systems, such as Redis [6], MongoDB [7], Cassandra [8], and Ceph [9], typically provide a replication feature. By maintaining multiple copies of data with the data systems, they provide high reliability, good availability, and rapid responsiveness to end users.

Table 1. Replication in distributed systems.

Data System	Number of Master	Number of Replica	Operation to Replica	Replication
Redis [6]	Single	Multiple	read	Async
Mongodb [7]	Single	Multiple	-	Async
HDFS [10]	Single	Multiple	read	Sync(1) Async(N-1)
Ceph [11]	Single	Multiple	-	Async
Cassandra [8]	Multiple	Multiple	read/write	Async
Dynamo [12]	Multiple	Multiple	read/write	Async

Table 1 summarizes the replication model employed in popular distributed data systems. The most popular replication model is the single–master and multiple–replica model, while a few distributed data systems, such as Dynamo [12] and Cassandra [8], allow multiple masters, preferring high availability to data consistency. In the single–master and multiple–replica model, all writes go through a master; the master reflects data changes to its dataset first and then propagates these changes to multiple replicas asynchronously. This mechanism makes it easy to coordinate the concurrent writes occurring with regard to the data without violating the consistency requirement.

We focus on the fact that the performance requirements are disparate across the data nodes of the system due to its asymmetric structure with replication. For master instances, rapid processing capability is expected because these instances are synchronous with user requests. In contrast, replica instances have a relaxed performance requirement in terms of latency because they have little impact on the user-facing performance. This property presents a possibility to gain a benefit by sacrificing the performance up to tolerable limits.

Motivated by this observation, this paper presents a novel DVFS algorithm called `Concerto`, which scales down the processor speed for replicas while having the processors in the master run at full speed. Modern processors manufactured by Intel, AMD, and ARM are equipped with the DVFS feature that enables the control of the processor speed within a certain range while using less power [13–15]. When the data service nodes are deployed in distributed data systems, `Concerto` sets the processor affinity separately for the master and the replica and throttles down the processor frequency for replicas. Considering that the number of replicas is typically twice more than the number of masters, CPU scaling down for replicas can lead to significant savings in power consumption.

Although `Concerto` appears to be straightforward conceptually, implementing it in practical systems and making an effective use of it are not without challenges. As a case study, we integrated `Concerto` into Redis, which is one of the most popular distributed in-memory NoSQL systems [6], though, when doing this, we face two major challenges. First, scaling down the processor speed of the replicas undesirably impairs the read performance associated with the replicas. Modern distributed data systems generally allow data reads from replicas to ensure proper load balancing, short latency, and high availability. As read requests are synchronous with users, it is unavoidable for the user-perceived latency to increase under `Concerto`. Second, the processing slowdown in replicas can increase the replication lag in distributed data systems. The replication lag refers to the time taken for the data changes of the master to be reflected in the replica. This synchronization delay

aggravates the staleness of the replicas, leading to greater data losses in the failure of the master.

We overcome these challenges as follows. To avoid unintended delays in the read latency, `Concerto` decouples the read and write path in Redis and processes them separately at the different cores, specifically processing reads at the high frequency and writes at the low frequency. Redis is natively designed with a single thread, making it challenging to realize the concurrent execution of reads and writes with it. We achieve this goal by identifying the part that takes a long time but has no dependency with writes in the read path and then offloading it to a different thread running at high speed. This compromise is disadvantageous in terms of energy savings, but it can prevent performance problems when using `Concerto`. Second, with regard to the replication lag, our in-depth analysis reveals that the scaling down of the processor in the replica has little impact on the replication lag. The master, responsible for several jobs to maintain data over the distributed environment, such as data propagation and/or replica synchronization, takes longer to process a write request than the replica. This performance gap allows room for the replica to operate at the lower speed without increasing the risk of a data loss in a failure.

We implemented the `Concerto` architecture in Redis 6.0.1. Performance evaluation with micro- and real-world benchmarks show that `Concerto` reduces the energy consumption of data systems by 32% on average and up to 51%, with a performance penalty of 3% compared to when it is not used. The remainder of this paper is organized as follows. Section 2 presents the background of this study, and Section 3 briefly highlights the related work. Section 4 details the design of `Concerto`. Section 5 evaluates `Concerto` under various workloads, and Section 6 discusses its applicability to different environments. Section 7 concludes the paper.

2. Background

In this section, we give a brief overview of dynamic voltage and frequency scaling (DVFS) and of the type of replication architecture used in distributed in-memory key-value store (Redis).

2.1. DVFS

DVFS is a well-known technique for reducing power and energy levels in electronic devices by adjusting the clock speed of their processors. The frequency at which a processor will run is proportional to the voltage ($f \propto V$) and the power the consumed by a main component varies linearly with $V^2 f$ [16]. Because the energy is the integral of the power over time and the time is inversely proportional to the frequency, energy consumption increases proportionally to the frequency squared ($E \propto f^2$). Hence, the processor can save a significant amount of energy by scaling down the clock speed. As an example, when it runs at half speed, the energy consumption is reduced to one fourth. Owing to its effectiveness, modern processors, from microprocessors to server processors, mostly support the DVFS functionality [17]. Below, we briefly explain the power-management technique adopted in Intel processors, our target architecture (Section 2.1.1), and the linux CPU frequency governors that enable one to set the desired frequency over the processors (Section 2.1.2).

2.1.1. Power Management of Intel Processors

With the rapid and continual increase in the popularity of both resource-limited embedded devices and energy-hungry data centers, the power consumption becomes an important concern in the design of computers. Intel processors provide two types of power management states: P-states and C-states [13]. P-states use different levels of voltages and frequencies of the CPU to allow optimization of the power consumption during runtime. Because lower voltage means the less power consumption, processors with P-states can save energy by throttling down the speed when a running task not in a computation-intensive state. Such states include the time waiting for the next chunk of data to process or the client's responses. In contrast to P-states, C-states are designed to reduce power

consumption in an idle mode. The C-state divides the sleeping modes into several levels. It determines the depth of the sleep mode appropriately while taking into account both the power consumption and the wake-up delay.

The proposed `Concerto` technique is associated with the runtime power management and is, thus, based on P-states. Table 2 shows the P-states of the Intel(R) Xeon Silver 4114 processor, which is the target processor here. The available frequencies are obtained using the `cpupower frequency-info` command in linux but the corresponding voltages are not provided; we estimated them for the purpose of this study using publicly available P-state examples [18] with the linear regression model implemented in scikit-learn [19].

Table 2. P-states in Intel(R) Xeon(R) Silver 4114.

Freq.	2.2 GHz	2.0	1.8	1.6	1.4	1.2	1.0
Vol.	1.10 V	1.06	1.02	0.98	0.94	0.90	0.86

2.1.2. Linux Governors

Most recent processors enable the user to set the frequency for each core individually within a specific range. CPU governors are components that determine at which frequency to operate the CPU within a given range. The current linux kernel is equipped with six types of `CPUFreq` governors [20]. The `performance` and `powersave` governors statically run the CPU at the maximum and minimum frequency, respectively within the corresponding ranges. The `userspace` governor allows user processes to adjust the frequency dynamically. The `ondemand` governor adjusts the processor speed according to the system load at a time window. The `conservative` governor is similar to the `ondemand` governor, but it changes the frequency more smoothly. Lastly, the `schedutil` governor sets the frequency more cooperatively with the scheduler. For example, it runs tasks assigned to the real-time scheduler statically at the highest frequency, while CPU throttling is applied only to the tasks executed in the CFS (Completely-Fair Scheduler). These governors can be chosen through the `sysfs` interface. In this paper, we use the `userspace` governor such that Redis can explicitly control the processor speed based on the disparate characteristics of instances.

2.2. Redis

Redis is an in-memory data store based on the key-value data model [6]. It supports various sets of data structures, including a hash table and sorted sets so as to satisfy many different tastes of clients. Furthermore, as Redis provides high availability and scalability via the built-in replication feature, it is finding widespread use in data centers.

Figure 1 shows the common architecture used when Redis is deployed. For good service robustness and availability, Redis uses data replication, which is realized by three types of instances: a master, a replica, and a sentinel. The master instance deals with all write requests of clients at the front end, maintaining an up-to-date dataset in distributed data systems. In contrast, replicas are asynchronously updated by the master, servicing read-only queries or simply providing backups for data safety. This asynchronous replication inevitably allows transient inconsistencies during certain time windows. Nevertheless, most distributed systems allow for such a defect, preferring high performance over strong consistency.

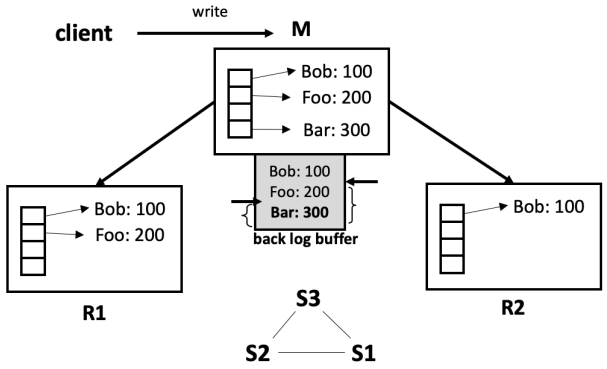


Figure 1. Redis architecture. M, R, and S stand for master, replica, and sentinel, respectively.

Figure 1 illustrates how a single master and two replicas cooperate in Redis. This example assumes that a client sends a `set` request with a key-value pair of `Bar` and 300 to the master. Upon the request, the master promptly updates its dataset structured in a hash table with new data and then transfers the request to the associated replicas. Subsequently, the master node immediately responds to the client without waiting for acknowledgement of data processing from replicas. The master node is notified asynchronously by the replicas regarding the amount of data processed in the replicas.

If the replica loses its connection with the master for some reason, such as a network issue or some other fault, the replica may significantly lag behind the master. To avoid data losses and inconsistencies, the replica performs partial synchronization when re-connected, receiving data differences from the master to ensure that the replica dataset is up-to-date. For this purpose, the master maintains a backlog buffer that holds all updates transferred to the replica in a fixed size of the in-memory buffer. The backlog buffer maintains an offset for each replica, indicating the amount of data processed for each replica. This offset proceeds upon the occurrence of an asynchronous acknowledgement by replicas. When a partial synchronization request arrives, the master identifies data differences by consulting the associated offset and transfers it to the replica.

The sentinel nodes are responsible for administration of the data system and are deployed to provide high availability without human intervention. As an example, the sentinel not only monitors the status of data service nodes and notifies the administrator if abnormalities are detected but also invokes the failover process to promote one of the replicas to the master whenever needed. By doing so, Redis is able to continue to offer their services even in the event of a failure.

3. Related Work

Our work enhances the energy efficiency of distributed data systems given with the judicious use of DVFS. In this section, we provide a brief overview of prior work that focused on power management in computing systems.

3.1. DVFS

Weiser et al. investigated the performance and potential energy saving of the strategies related to dynamic control of the clock speed in response to application activities [21]. They showed that a final-grained dynamic adjustment of the CPU clock can reduce energy consumption by up to 70% with limited impact on performance. Similarly, Pering et al. also examined the effectiveness of dynamic voltage scaling (DVS), but their study was conducted with benchmarks on actual product [22]. They also observed that DVS algorithms reduce the energy usage by 46%, though a careful choice of the parameters is necessary. To go one step further, Govil et al. explored sophisticated dynamic speed-setting algorithms in an effort to realize a better tradeoff between the power and the delay [23]. Shieh et al. further optimized a DVFS algorithm by taking the voltage transition overhead into consideration [24]. They constructed a linear programming model considering the transition

overhead and adjusted the CPU voltage dynamically such that it minimizes the energy consumption while maintaining the deadline miss rate.

DVFS has been actively studied in the realm of real-time systems. Aydin et al. dynamically adjusted the CPU frequency to reduce the energy consumption, while satisfying the deadlines of real-time tasks with large variations of the execution time [25]. They monitor task execution and reschedule unused CPU time to be used by other tasks. Jejurikar presented a method of distributing the slack time of real-time tasks to maximize the energy savings by extending idle intervals [26]. Kwon et al. proposed an optimal voltage allocation technique that minimizes the total energy consumption when tasks with different arrival times and deadlines are executed over multiple voltages [27]. Other researchers [28–30] extended dynamic voltage scheduling for real-time applications on multi-core processors.

3.2. Power Management for Data Centers

Several studies have focused on power management in large-scale data centers. Tolia et al. identified that the CPU no longer accounts for the majority of the energy consumption on servers and presented a virtual machine consolidation concept that can be used along with DVFS. By migrating virtual machines off to the under-utilized machines, they enable more idle machines to be turned off, leading to great energy savings in data centers [31]. Raghavendra et al. presented a method to coordinate individual power management approaches in data centers so as to prevent them from interfering with each other in unexpected ways and, thus, to realize overall energy savings [32]. Chen et al. further elaborated on power management techniques for computer systems by considering that frequent on-off activities can accelerate the wear-out of internal components, resulting in additional costs for repair and replacement [33].

Although a number of studies have been conducted to reduce the energy consumed by computer systems, there is no prominent study that exploits the disparate requirements of data nodes in distributed systems for this goal. The need to take advantage of the performance relaxation in replicas for other benefits arises in the non-academic literature. The Redis community advises that the AOF (Append-Only-File) feature be disabled in the master node, which logs updates into persistent storage, as this aggravates the burdens of the master node [6]. Ceph supports the configuration of storage devices differently depending on the requirements of the application [34] and/or its functionality [35]. As an example, in relation to this method, the concept of device classes was recently introduced, with user level interfaces provided with which users can deploy the best-fitting storage devices for their objectives [34]. By making use of this approach, Ceph users can assign state-of-the-art storage devices, such as SSDs, to the primary data node while using massive and inexpensive storage devices, like HDDs, for the remainder of the data nodes [36,37]. These approaches and our work have similar motivations, but they were introduced at different levels. To the best of our best knowledge, `Concerto` is the first approach to consider clock-speed scaling from the viewpoint of the distributed systems, focusing on heterogeneous performance constraints across data nodes.

4. Design of Concerto

In this section, we describe `Concerto` in detail. To reinforce the practical aspect of the proposed technique, we explain `Concerto` in the context of Redis in the form of a case study.

4.1. Overview

`Concerto` as presented here takes advantage of the fact that performance constraints are disparate across instances in distributed systems. Master instances have a great impact on user-perceived performance and, thus, necessitate high data processing capabilities. In contrast, as replica instances act as a substitute for the original corresponding dataset in a sudden failure of the master, they scarcely affect the user experience during normal operations. Based on this property, we present the novel DVFS policy called `Concerto`,

which deliberately scales down the processor speed when they are used to process replicas. This policy can allow for energy savings with a minimal loss of performance.

The Concerto power manager configures the kernel to use the userspace governor to control the processor speed as the governor allows a user process to adjust the processor speed as needed. In our implementation, the power manager associates each instance with the specific core on deployment time and shares the information with them. When an instance starts running, it figures out whether it is a master or a replica in distributed data systems, and it sets the frequency of associated core accordingly by making use of the sysfs interface. The sysfs is the pseudo-file system interface provided by linux kernel to adjust the frequency of CPU. By writing the proper number into the virtual file (cpufreq/scaling_mad_freq) internally linked with the processor frequency modulator, the user applications sets the processor speed as desired. Once the processor frequency is all set, the instance sets its CPU affinity to the corresponding core using the sched_setaffinity library [38]. By doing so, Concerto executes the master for the core to run at full speed and the replica for the core to run at slow speed.

This architecture makes sense because modern processors typically have several cores and allow separate adjustments of the P-state levels for each core. In this study, the fast cores for the masters run at the full speed (2.20 GHz), while the cores assigned to the replicas are forcefully scaled down to a lower speed (1.0 GHz). The Concerto power manager automatically detects dynamic changes of the processor speed and resets the processor affinity of the data nodes as necessary.

4.2. Read Path Offloading

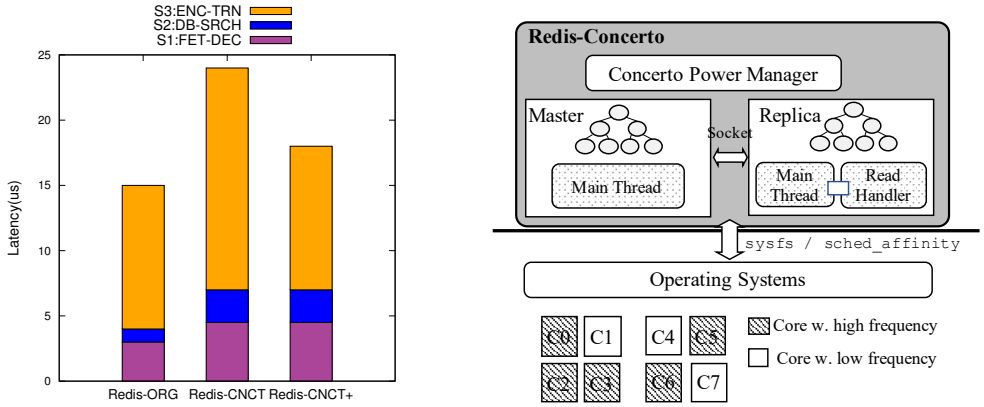
One primary intention for replication is to reduce the read latency. By servicing data from a node geographically close to the user, data systems can achieve low latency for reads. For this reason, most distributed data systems allow read operations to replicas, while granting writes only for the master. In Concerto, read queries sent to replicas can experience a performance penalty owing to the scaling down of the processor.

One possible approach by which to mitigate this shortcoming is by adaptive processor throttling. By temporarily increasing the processor speed for read operations, we can prevent the read latency from increasing due to DVFS. This mechanism has an advantage in that it can be applied to single-threaded data systems, such as Redis. However, changing the processor speed frequently for every read query imposes excessive overhead. It not only requires the software overhead to configure the speed through the virtual file system but also adds transition overhead when changing the CPU frequency. This cost outweighs the benefit of processing read queries at high speed.

Based on this observation, Concerto resolves this problem by separating the read and write paths and executing the read operation with a thread running at full speed. To this end, Concerto selectively scales down a subset of the cores and forces the read path to run on the fast-running core. However, realizing concurrent reads and writes in the single-threaded data systems comes at another cost. It significantly increases the code complexity required to resolve the conflict between the concurrent reads and writes. In addition, employing a weaker consistency model, such as snapshot isolation, is inevitable, leading to a decrease of the consistency level of the data system.

To strike a balance between the cost and benefit, we present a partial offloading scheme. This scheme processes the part that requires complex coordination between reads and writes using a single thread called MainThread, while offloading the part that can be executed independently in the read path to another thread, called ReadHandler. To this end, we investigate which part of the read path can be offloaded in Redis. Figure 2a provides a breakdown of the read latency for the three steps of the request in Redis. Redis-ORG and Redis-CNCT show the read latency when executing the replica at a high frequency and at a low frequency, respectively. In the first step, Redis records the arrival of a request at the socket and fetches it from the kernel buffer, after which decoding takes place (S1:FET-DEC). If the request is distinguished as a read request, Redis searches the hash table using

the given key to retrieve the address of the associated value (S2:DB-SRCH). Lastly, the retrieved value is encoded using the Redis protocol and transferred to the client over a socket communication (S3:ENC-TRN).



(**a**) Read latency breakdown for replica. (**b**) Architecture of Redis with `Concerto`.

Figure 2. Read latency analysis and overall architecture of Redis with Concerto.

The first two steps are not appropriate for offloading. In the first step, Redis cannot determine the type of request. The second step requires access to the database, which is the critical section for concurrent reads and writes. In contrast, the last step can be offloaded because it has no dependency with other write operations. Although the value is updated during the `ReadHandler` access step, Redis updates the value in an out-of-place manner; thus, there is no contention between threads to access the value. Moreover, Figure 2a shows that the last step accounts for a large portion of the read latency and, thus, incurs the most significant delay when the processor scales down. Here, 67% of the delay occurs during the process of encoding and transferring the data. Based on this observation, `Concerto` offloads it to `ReadHandler` such that encoding and transferring value to the end user can be done at full speed. For a fluid execution of the read operation, we use a lock-free queue for inter-thread communication. `MainThread` enqueues the request into the lock-free queue when the value is retrieved from database and wakes `ReadHandler` to process it. Consequently, as shown in Figure 2a, the increase in a read latency caused by the scaling down of the processor is reduced to one third with read-path offloading (`Redis-CNCT+`). Figure 2b depicts the overall architecture of Redis with `Concerto`.

4.3. Replication Lag Analysis

The replication lag refers to the delay between a write occurring to the master and being reflected on the replica. When the replica has long write latency, the replication lag inevitably increases, leading to the provision of more stale data in the replica and increasing the risk of data losses in the failure. In addition, a long replication lag indicates to the master that a problem occurs in data transmission, causing it to re-transmit data to the replicas. This is termed partial synchronization, and it unnecessarily increases both the master-node burden and the network traffic.

However, our experimental analysis shows that it is unlikely for the above situation to occur in practice. Figure 3 shows the propagation rate of the master and the data processing time in the replica with and without the scaling down of the processor. This result is obtained by measuring the duration of each component in Redis, running the memtier benchmark with one million `SET` operations. Each operation has a 16 B key and a 1 KB value. We run Redis with a single master with two replicas. The numbers are the averages of the overall results. In the figure, we find that the processing speed of the replica greatly exceeds the propagation rate even when the processor scales down. The write request from the master arrives once every 77.69 us on average, while the corresponding processing time takes 10.98 us and 22.28 us on the replica when the processor runs at full speed and when

it scales down, respectively. Because the master is responsible for more functions than the replica, such as responding to client and synchronizing replicas, it has much longer processing time per request than the replica.

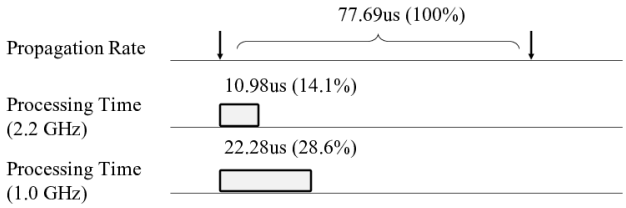


Figure 3. Comparison of the propagation and processing rates of a write operation.

Figure 4 shows the write path of the master and of the replica, respectively. The master initially reads the data from the kernel buffer to the query buffer and applies it to the database. Subsequently, it maintains the data within the two buffers: backlog buffer and propagate buffer. They are needed to propagate the data safely to the replicas. In contrast, the replica only performs the first two steps. As a result, the master undertakes data copying six times in total, while the replica does this only three times. For this reason, the propagation rate of the master lags the processing speed of the replica, even when the replica runs on the scaled-down processor. This outcome shows that, although the write latency increases by nearly two times, it can be hidden because write processing is finished before the next request.

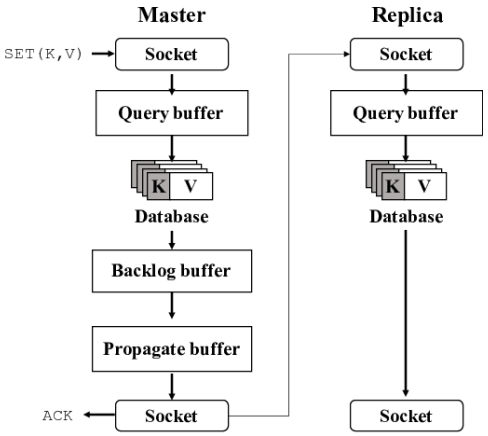


Figure 4. Write path of master and replica in Redis.

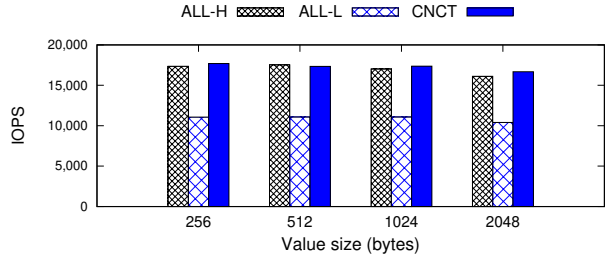
5. Performance Evaluation

To evaluate the effectiveness of `Concerto`, we implemented it on Redis 6.0.1. We performed experiments using a prototype on an Intel(R) Xeon(R) Silver 4114 processor with ten cores. Table 2 shows the P-states available in the processor. The cores assigned to the master run at the full speed (2.20 GHz), while the cores mapped to the replicas are scaled down to the lower speed (1.0 GHz). For the experiments, we deployed a single master and two replicas with Redis, communicating via a socket connection on the same physical machine. This configuration minimizes the effect of any network delay on the performance measurement, highlighting the strength and drawbacks of `Concerto`. Note that this setting shows the lower bound of the performance with `Concerto` because its impact on the performance decreases as the network delay increases. We investigate the performance and energy tradeoff of `Concerto` using the Memtier benchmark [39] and the YCSB benchmark [40].

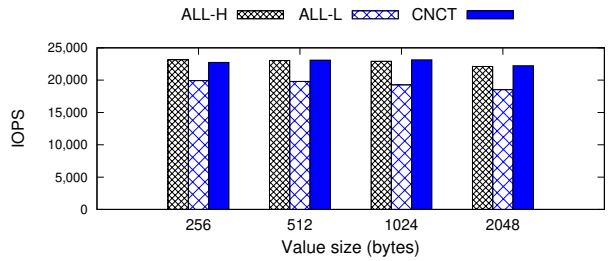
5.1. Benchmark through the Master

Figures 5 and 6 show the IOPS and energy consumption outcomes of Redis when running the Memtier benchmark. We generate one million `SET` and `GET` operations varying

the value size from 256 B to 2 KB. In the figures, we compare the performance and energy consumption of `Concerto` (`CNCT`) with those when executing all nodes at a high frequency (`ALL-H`) and at the low frequency (`ALL-L`). Figure 6 presents the energy consumed by the processor during the execution of the benchmark; this value is calculated by multiplying the power and the total elapsed time for each configuration. For comparison, we plot the energy consumption for two different cases: when a master runs with a single replica (`M-R`) and with two replicas (`M-2R`). This is done in an effort to observe the energy consumption trend as the replication factor increases.



(**a**) SET



(**b**) GET (Master)

Figure 5. IOPS with Memtier benchmark.

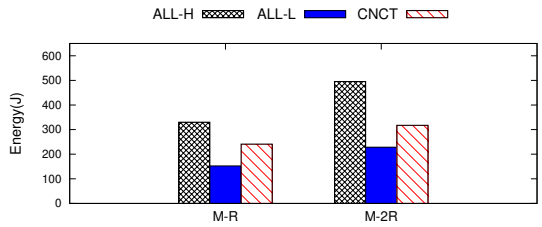


Figure 6. Energy consumption with Memtier benchmark.

With respect to energy efficiency, `ALL-L` achieves the most significant savings, reducing the energy consumption by 54% compared to `ALL-H`. However, it inhibits the performance significantly, decreasing IOPS by 36% and 15% on average for `SET` and `GET` operations compared to `ALL-H`. In contrast, `CNCT` decreases the energy consumption by 27–36%, but this is achieved while maintaining the performance. `CNCT` provides throughput identical to that of `ALL-H` for both `SET` and `GET` operations.

We investigate the effectiveness of `Concerto` using YCSB benchmarks. The YCSB benchmark suite has six different workloads, and Table 3 summarizes the characteristics of each. We initially load one million data items and perform one million operations for each workload. Figures 7 and 8 show the IOPS and the energy consumption outcomes for the YCSB benchmark. While `ALL-L` performs worse than `ALL-H` by up to 25% (20% on average), `Concerto` achieves a performance similar as `ALL-H` across all YCSB workloads. This experimental result demonstrates that the performance relaxation in replicas can be hidden to from the user experience, thus enabling considerable energy savings without a noticeable performance loss.

Table 3. YCSB workload characteristics.

WK	Description	Ratio
Load	Initial database establishment	100% Insert
A	Session store recording actions	50% Read, 50% Update
B	Photo tagging and tag reads	95% Read, 5% Update
C	User profile cache	100% Read
D	User status update	95% Read, 5% Insert
E	Recent post scan in conversation	95% Scan, 5% Insert
F	Database	50% Read, 25% Read-Modify-Write, 25% Update

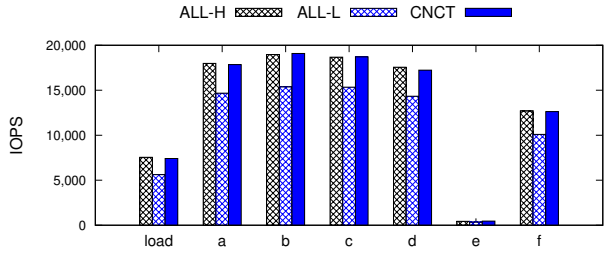


Figure 7. IOPS with YCSB.

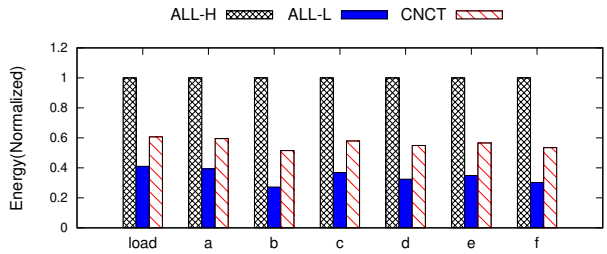
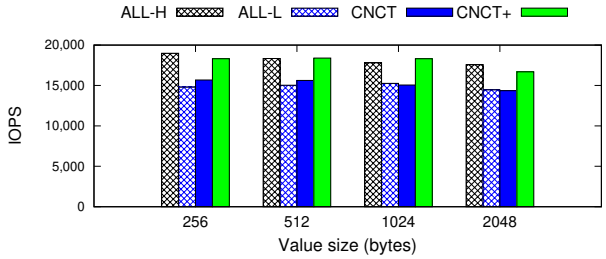


Figure 8. Energy consumption with YCSB.

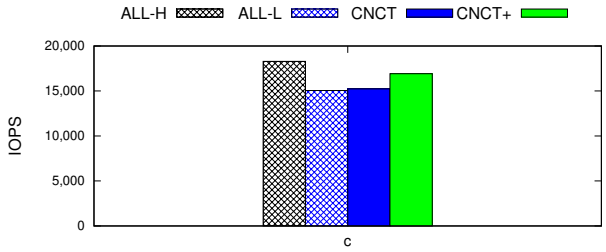
With regard to energy consumption, `Concerto` is inferior to `ALL-L` because we stick to executing the master node at full speed. `Concerto` reduces energy consumption by 44% on average compared to `AlLL-H`, whereas `ALL-L` uses less energy than `ALL-H` by 67% on average. However, the advantage of `ALL-L` in terms of energy is largely compensated for by its performance loss, making it less feasible in practice.

5.2. Benchmark through a Replica

The master–replica model typically allows the replica to accept the read queries, thereby enhancing the availability and performance capabilities of data service platforms. This characteristic is at odds with the concept of `Concerto` because scaling down the replica can deliver a decrease in the user experience.`Concerto` is augmented to offload the independent part of the read path to the fast-running thread. To evaluate the efficacy of the optimization process, we measured performance by reading data from the replica. Figure 9 shows the IOPS of Memtier and the YCSB benchmarks when sending the read requests to the replica. For YCSB, we only report the result of workload C because the others include write requests, and, they, therefore, fail when directed to the replica. We study two versions of `Concerto`: `CNCT` processes all requests (e.g., read requests from users and write requests from master) with a single thread, and `CNCT+` performs the offloading optimization for read requests. `CNCT` generates 20% of the performance loss compared to `ALL-H`, which is identical to the outcome of `ALL-L`. However, `CNCT+` significantly reduces the performance gap with the partial offloading technique, which is only 3% on average. Considering that this measurement is performed under the high-bandwidth network configuration, this performance penalty will be negligible in practice.



(**a**) GET (Memtier)



(**b**) Workload C (YCSB)

Figure 9. Read performance for replica.

6. Discussion

Although our study mostly focuses on the Intel Xeon processor and Redis, it can be generally applied to different architectures and applications. With regard to the underlying processors, AMD uses a power management technology called `AMD PowerNow!`, in which fine and dynamic control of the voltage and frequency of the cores is possible [14]. Similarly, ARM processors, which have been tailored mostly for embedded systems powered by batteries but are now seriously considered for use in data centers, provide various built-in hardware design methods to reduce power usage levels [15]. They not only support the dynamic on-off switching of the cores (HotPlug) but also allow for modifications of the clock speed according to the computation requirements (DVFS). Furthermore, Amazon EC2 also supports processor state control (C-states/P-states) of the underlying physical machine for the several specific types of instances [41]. These processors all empower an application to optimize power and performance to best satisfy the user's requirements. With regard to applications, the relaxation of performance requirement for replicas is not unique in Redis. Distributed data systems that asynchronously maintain replicas, such as Mongodb [7], HDFS [10], and Ceph [9], share this property. Taking into account all of these aspects, we argue that our approach is timely and applicable in a wide range of circumstances, although the implementation methodologies may vary across different systems.

7. Conclusions

This paper presents a novel power management technique for distributed data systems called `Concerto`. `Concerto` exploits relaxed performance constraints for replicas and reduces the power consumption by scaling down the processor speed assigned to the components. The performance penalty that can occur during read operations on a replica is prevented through partial read-path offloading. `Concerto` was implemented in Redis 6.0.1, and a performance evaluation using a prototype shows that `Concerto` substantially reduces the CPU energy with a limited impact on the performance.

Author Contributions: Conceptualization, J.L. and E.L.; methodology, J.L. and E.L; software, J.L.; validation, J.L. and E.L.; investigation, E.L.; writing—original draft preparation, E.L.; writing—review and editing, E.L.; visualization, J.L. and E.L; supervision, E.L.; project administration, E.L.; funding acquisition, E.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Basic Science Research Program through the National Research Foundation of Korea (NRF-2019R1A2C1090337).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Shehabi, A.; Smith, S.; Sartor, D.; Brown, R.; Herrlin, M.; Koomey, J.; Masanet, E.; Horner, N.; Azevedo, I.; Lintner, W. *United States Data Center Energy Usage Report*; Technical Report; Lawrence Berkeley National Lab.(LBNL): Berkeley, CA, USA, 2016.
2. Masanet, E.; Shehabi, A.; Lei, N.; Smith, S.; Koomey, J. Recalibrating global data center energy-use estimates. *Science* **2020**, *367*, 984–986. [CrossRef] [PubMed]
3. Bianchini, R.; Rajamony, R. Power and energy management for server systems. *Computer* **2004**, *37*, 68–76. [CrossRef]
4. David, H.; Fallin, C.; Gorbatov, E.; Hanebutte, U.R.; Mutlu, O. Memory power management via dynamic voltage/frequency scaling. In Proceedings of the 8th ACM International Conference on Autonomic Computing, Karlsruhe, Germany, 2011; pp. 31–40.
5. Brown, R.E.; Brown, R.; Masanet, E.; Nordman, B.; Tschudi, B.; Shehabi, A.; Stanley, J.; Koomey, J.; Sartor, D.; Chan, P.; et al. *Report to Congress on Server and Data Center Energy Efficiency: Public Law 109-431*; Technical Report; Lawrence Berkeley National Lab. (LBNL): Berkeley, CA, USA, 2007.
6. Redis Labs. Redis Replication. 2020. Available online: https://redis.io/topics/replication (accessed on 20 June 2021).
7. MongoDB. Replication. 2020. Available online: https://docs.mongodb.com/manual/replication/ (accessed on 20 June 2021).
8. Cassandra, A. Multi-Master Replication: Versioned Data and Tunable Consistency. 2020. Available online: https://cassandra.apache.org/doc/latest/architecture (accessed on 20 June 2021).
9. Ceph. Ceph Architecture. 2020. Available online: https://docs.ceph.com/en/latest/architecture/ (accessed on 20 June 2021).
10. Apache. Data Replication in HDFS. 2020. Available online: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html (accessed on 20 June 2021).
11. RedHat. Ceph Replication. 2020. Available online: https://access.redhat.com/documentation/en-us/red_hat_ceph_storage/4/html/architecture_guide/the-core-ceph-components (accessed on 20 June 2021).
12. DeCandia, G.; Hastorun, D.; Jampani, M.; Kakulapati, G.; Lakshman, A.; Pilchin, A.; Sivasubramanian, S.; Vosshall, P.; Vogels, W. Dynamo: Amazon's highly available key-value store. *ACM SIGOPS Oper. Syst. Rev.* **2007**, *41*, 205–220. [CrossRef]
13. Intel. Intel Xeon Processor E3-1200 v5 Product Family. 2015. Available online: https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/xeon-e3-1200v5-vol-1-datasheet.pdf (accessed on 20 June 2021).
14. AMD. AMD PowerNow Technology. 2020. Available online: https://www.amd.com/system/files/TechDocs/24404a.pdf (accessed on 20 June 2021).
15. ARM. ARM Cortex-A Series Programmer's Guide. Power Management. 2020. Available online: https://developer.arm.com/documentation/den0013/d/Power-Management (accessed on 20 June 2021).
16. Intel Corporation Enhanced SpeedStep® Technology for the Intel® Pentium® M Processor White Paper, March 2004. In *Recovered 30/1/2011 from World Wide Web*; Tech. Rep.; Intel Corporation: Santa Clara, CA, USA, 2004.
17. Wikichip. Zen—Microarchitectures AMD. 2020. Available online: https://en.wikichip.org/wiki/amd/microarchitectures/zen (accessed on 20 June 2021).
18. WikiChip. Frequency Behavior Intel. 2020. Available online: https://en.wikichip.org/wiki/intel/frequency_behavior (accessed on 20 June 2021).
19. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
20. Linux. Linux CPUFreq Governors. 2020. Available online: https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt (accessed on 20 June 2021).
21. Weiser, M.; Welch, B.; Demers, A.; Shenker, S. Scheduling for reduced CPU energy. In *Mobile Computing*; Springer: Boston, MA, USA, 1994; pp. 449–471.
22. Pering, T.; Burd, T.; Brodersen, R. The simulation and evaluation of dynamic voltage scaling algorithms. In Proceedings of the 1998 International Symposium on Low Power Electronics and Design (IEEE Cat. No. 98TH8379), Monterey, CA, USA, 10–12 August 1998; pp. 76–81.
23. Govil, K.; Chan, E.; Wasserman, H. Comparing algorithm for dynamic speed-setting of a low-power CPU. In Proceedings of the 1st Annual International Conference on Mobile Computing and Networking, Berkeley, CA, USA, 13–15 November 1995; pp. 13–25.

24. Shieh, W.Y.; Pong, C.C. Energy and transition-aware runtime task scheduling for multicore processors. *J. Parallel Distrib. Comput.* **2013**, *73*, 1225–1238. [CrossRef]
25. Aydin, H.; Melhem, R.; Mossé, D.; Mejía-Alvarez, P. Power-aware scheduling for periodic real-time tasks. *IEEE Trans. Comput.* **2004**, *53*, 584–600. [CrossRef]
26. Jejurikar, R.; Gupta, R. Dynamic slack reclamation with procrastination scheduling in real-time embedded systems. In Proceedings of the 42nd Design Automation Conference, Anaheim, CA, USA, 13–17 June 2005; pp. 111–116.
27. Kwon, W.C.; Kim, T. Optimal voltage allocation techniques for dynamically variable voltage processors. *ACM Trans. Embed. Comput. Syst. TECS* **2005**, *4*, 211–230. [CrossRef]
28. Lorch, J.R.; Smith, A.J. Improving dynamic voltage scaling algorithms with PACE. *ACM SIGMETRICS Perform. Eval. Rev.* **2001**, *29*, 50–61. [CrossRef]
29. Seo, E.; Jeong, J.; Park, S.; Lee, J. Energy efficient scheduling of real-time tasks on multicore processors. *IEEE Trans. Parallel Distrib. Syst.* **2008**, *19*, 1540–1552.
30. Chen, J.J.; Yang, C.Y.; Kuo, T.W.; Shih, C.S. Energy-efficient real-time task scheduling in multiprocessor dvs systems. In Proceedings of the 2007 Asia and South Pacific Design Automation Conference, Yokohama, Japan, 23–26 January 2007; pp. 342–349.
31. Tolia, N.; Wang, Z.; Marwah, M.; Bash, C.; Ranganathan, P.; Zhu, X. Delivering energy proportionality with non energy-proportional systems-optimizing the ensemble. *HotPower* **2008**, *8*, 2.
32. Raghavendra, R.; Ranganathan, P.; Talwar, V.; Wang, Z.; Zhu, X. No "power" struggles: Coordinated multi-level power management for the data center. In Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems, Seattle, WA, USA, 1–5 March 2008; pp. 48–59.
33. Chen, Y.; Das, A.; Qin, W.; Sivasubramaniam, A.; Wang, Q.; Gautam, N. Managing server energy and operational costs in hosting centers. In Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, Banff, AB, Canada, 6–10 June 2005; pp. 303–314.
34. Hat, R. Ceph: Get the Best of Your SSD with Primary Affinity. 2018. Available online: https://access.redhat.com/documentation/en-us/red_hat_ceph_storage/1.3/html-single/storage_strategies_guide/index (accessed on 20 June 2021).
35. Ceph. Hardware Recommendations. 2016. Available online: https://docs.ceph.com/en/latest/start/hardware-recommendations/ (accessed on 20 June 2021).
36. Ceph. Ceph: Get the Best of Your SSD with Primary Affinity. 2015. Available online: https://ceph.io/geen-categorie/ceph-get-the-best-of-your-ssd-with-primary-affinity/ (accessed on 20 June 2021).
37. Guyot, M. Ceph Hybrid Storage Tiers. 2017. Available online: https://ceph.io/planet/ceph-hybrid-storage-tiers/ (accessed on 20 June 2021).
38. Linux. Sched Setaffinity(2)-Linux Manual Page. 2021. Available online: https://man7.org/linux/man-pages/man2/sched_setaffinity.2.html (accessed on 20 June 2021).
39. Redis. Memtier Benchmark. 2020. Available online: https://github.com/RedisLabs/memtier_benchmark (accessed on 20 June 2021).
40. Cooper, B.F.; Silberstein, A.; Tam, E.; Ramakrishnan, R.; Sears, R. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing*; Association for Computing Machinery: New York, NY, USA, 2010; pp. 143–154. [CrossRef]
41. Amazon. Processor State Control for Your EC2 Instance. 2021. Available online: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/processor_state_control.html (accessed on 20 June 2021).