Heat-aware Load Balancing - Is it a Thing?

Lukas Iffländer, Norbert Schmitt, Andreas Knapp, Samuel Kounev
{firstname}.{lastname}@uni-wuerzburg.de
University of Würzburg

Abstract

Dynamic frequency scaling, also known by the name of its most common implementation form Intel "Turbo Boost," has been around for over ten years. While it provides a short time boost to a CPU's clock rate, it has no permanent influence on it. Existing work either tried to characterize the boost's behavior or explicitly disabled the boost not to influence their performance models. We present heat-aware load balancing. This approach allows migrating a service between servers in a matter that keeps the boosted state active as long as possible. We introduce a prototype implementation that shows the feasibility of our approach in a simulated environment.

1 Introduction

Modern processors can exceed their designed clock rate for short time frames. Starting with Intel's Turbo Boost technology in 2008, these capabilities have evolved, and competing CPU manufacturers have adopted these technologies.

At first, the idea was to increase only the clock rate of a single Central Processing Unit (CPU) core. When first introducing Turbo Boost in 2008, multi-core CPUs were expanding out of the enthusiast market and became available to the general public. At the same time, many applications—especially video games—were single-threaded. To increase the performance of these applications, all cores but one could be disabled to allow for a permanent increase in the remaining core's clock rate.

In addition to continually boosting a single core or few cores, Intel added the capability to exceed a CPU's thermal budget temporarily. This boost can be between around 20% (for servers) and over 100% (for ultra low power CPUs, e.g., Intel's Y-Series).

Related work mostly deals with the characteristics of Turbo Boost [2] and its influence on specific computing scenarios, e.g., high-performance computing [4]. Many works dealing with software performance regard this feature (similar to HyperThreading) as an unwelcome interference to their performance models and disable this feature. While this is a sensible choice to validate performance models, it limits their real-life applicability.

Instead of seeing Turbo Boost as a nuisance, in this work, we work on harvesting its potential. Imagine an infrastructure with several homogenous compute nodes. On each of these nodes, an application is running, consuming around 60% of that nodes compute resources. We assume each of these applications is already consolidated as much as possible. Now we want to add a single application that would require 50% of the resources of one of our hosts. Within the existing possibilities, we would have to boot up another host machine (or violate an SLA on purpose).

As we have learned above, Turbo Boost allows a Server to have around a 20% higher clock rate than usual. Thus, on a boosted server, we could deploy the new application as long as the boost stays up. Unfortunately, the boost does not stay up infinitely. Here the idea of heat-aware load balancing enters the stage. When a host drops out of boost (or is predicted to drop out of boost), we migrate the application to the host, expected to stay boosted for the longest possible time. We continue this process as long as the applications require more resources than available.

We expect this approach to provide a constant boost in large environments where there is always a boostable server available. For smaller settings, this approach could help to quickly deploy the application and then later migrate it to a newly booted additional host. The contributions of our work are **(1)** an approach to heat-aware load balancing and **(2)** a prototype implementation using software-defined networking, including an initial evaluation.

The remainder of this work first introduces Turbo-Boost in Section 2, and our approach to utilize it for increased performance in Section 3. Next, we implement it in Section 4 and perform the first evaluation of its feasibility in Section 5. Last, Section 6 concludes the paper and gives an outlook on future work.

2 Intel TurboBoost

Compared to classical overclocking, the CPU is still working within thermal design specifications, the amount of heat a chip is specified to emit, and the cooling system can dissipate. Still, it can exceed for a short period with a higher clock speed than usual. The technology is independent of the Operating System (OS) and enabled by default. It can only be disabled in BIOS and is only CPU-based, not core-based. Between version 1.0 and version 2.0 of the boost technology, the principle of operation did not change a lot,

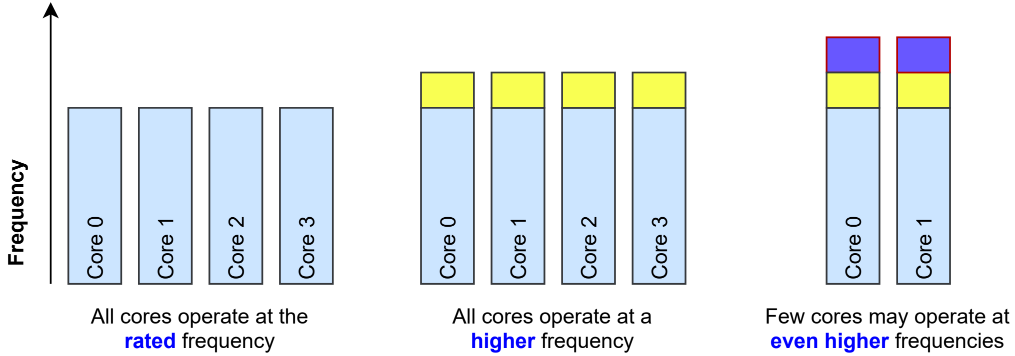


Figure 1: Visualization of turbo boost [1]

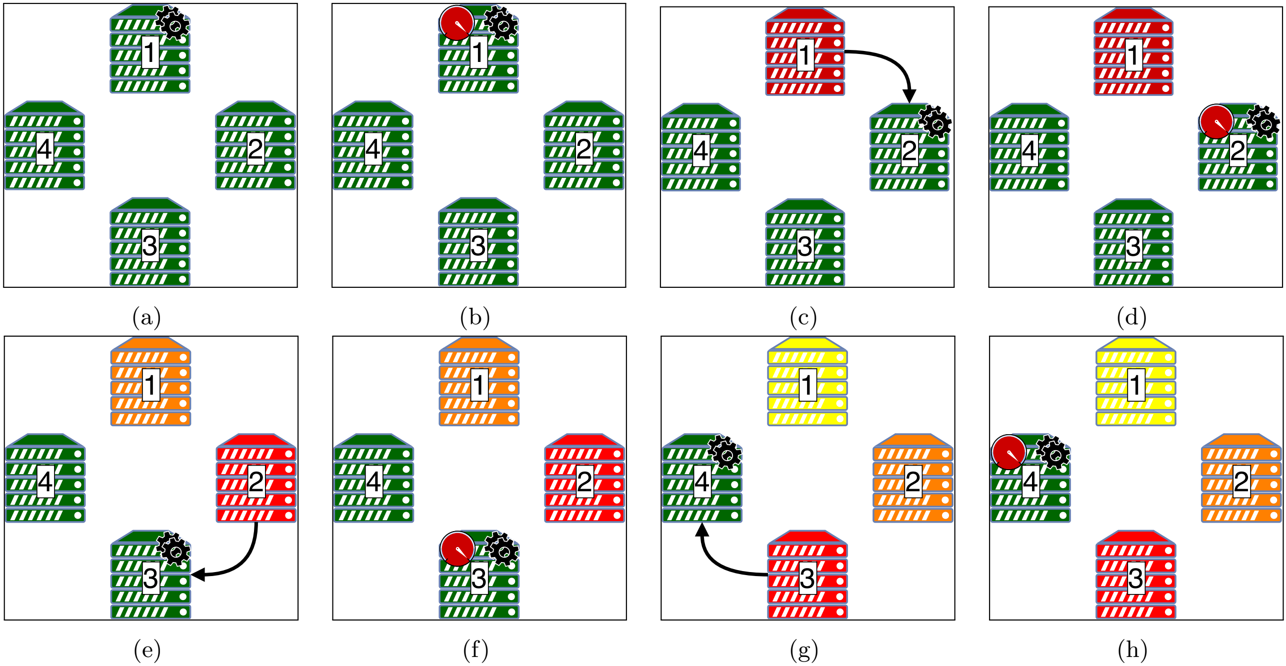


Figure 2: Overview of a full rotation using heat-aware load balancing.

but it was improved. Figure 1 shows how the boost works.

3 Approach

Our approach follows the idea of having an application, always running on a boosted server. Figure 2 shows an example behavior for a set of four servers. We assume that either the application alone creates enough load to trigger the server going into boost mode or that the server already has a high enough load that the new application can only fit when boosting the CPU frequency.

We deploy a new application on the first host in Figure 2a. This activity leads to the host's CPU switching to boost mode in Figure 2b. The boost mode results in a higher creation of heat than the cooling solution can transport away. Thus, after some time, the thermal budget of the CPU is exhausted (shown by the server turning red), and the CPU leaves its boost mode in Figure 2c. We simultaneously move the application to the next server to keep it on a machine at an elevated clock rate. Thus, this target server now enters its boosted state (Figure 2d). Again, at some point, the server exits its boosted state, and we move the application to the next server (Figure 2e). During the time the application spent on the second server, the first server cools down a little (represented by the server turning orange). The process repeats with boosting (Figure 2f), moving the fourth and last server, and the first and second server again cooling down (Figure 2g), and also with boosting the last server (Figure 2h). As shown by the heat icon in the figures, the accumulated heat level on the hosts diminishes after exiting boost state and relinquishing the application to another host. At some point, the first host again is capable of going into boost state (represented by it turning green too). When the last host leaves its boost state, we assume that the first

host is ready to boost again. Thus, the application once more moves to the first host. From thereon, it continues through the rotation.

For the implementation of the heat-aware load balancing, we propose to use Software-defined Networking (SDN). SDN allows for the dynamic modification of network packets. Here, we use SDN to change the destination of requests to a service based on the machine's temperature and ability to go into or stay in boosted mode.

Therefore, we make a list of computing machines known as workers. Our application runs on each of these machines and listens to the active port. The application does not require a significant amount of resources when running without receiving queries.

4 Implementation

Our application has multiple components that interact: 1. a central monitoring component, 2. distributed worker-side monitoring components, 3. an SDN controller, and 4. an SDN-enabled switch.

While the worker-side components must run on the worker machines, the other parts can run on a single device or spread over multiple machines.

Central Monitoring The central monitoring component collects data from the worker machines. For this task, our choice fell to InfluxDB, and Chronograph visualizes the recorded data.

Worker-side Monitoring To collect information regarding CPU frequency and temperature from the worker machines, we use Telegraf.

SDN Controller We use Ryu as an SDN controller. Ryu is lightweight, supports basic switching, a REST interface, and provides for a simplified extension using simple Python scripts.

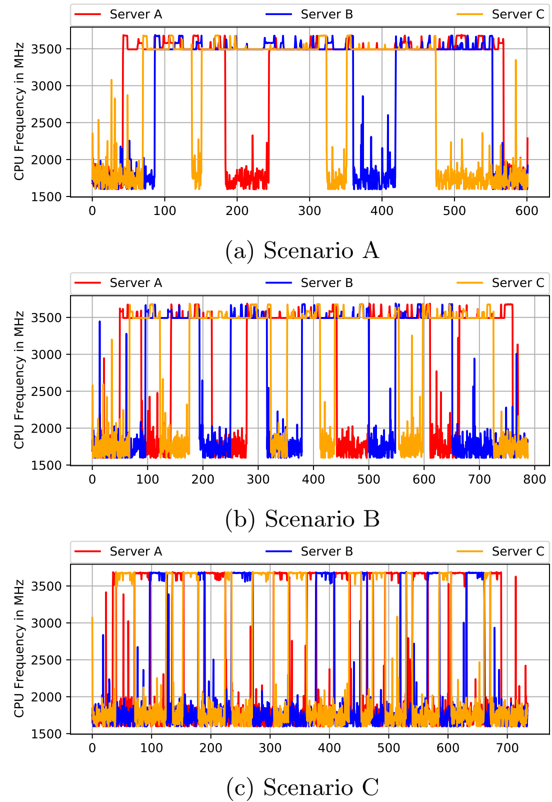
SDN Switch In general, any OpenFlow (OF) 1.3 compatible switch should suffice for our approach. To ensure full OF compliance, we use Open vSwitch.

5 Evaluation

We put our system under load using the LU workload. We deploy the LU workload in the version from SPEC SERT [5] reimplemented in BUNGEE [3].

Figure 3 shows the CPU frequency development on three servers with our load balancer enabled throughout three different LU workloads, that extremely overbook Figure 3a, slightly overbook Figure 3b, and do not entirely exceed the system resources Figure 3c.

The regular maximum frequency for the used CPUs is 3.5 GHz. The figure shows that the amount of time spent at higher clock rates increases with decreasing total system load up to a point in Scenario C, where the active CPU is always in a boosted state. Thus, overall our approach of keeping the active server at maximum frequency is feasible.



(a) Scenario A

(b) Scenario B

(c) Scenario C

Figure 3: Highest core frequency during the experiment.

6 Conclusion and Future Work

In this work, we introduced heat-aware load balancing. We described the underlying idea of short term CPU frequency scaling and our approach to using it for increased performance. We provide an implementation and the first evaluation of this prototype. The results show that for a matching workload, our approach achieves its goals.

In future work, we will extend our evaluation to performance metrics like throughput and latency and perform temperature and power consumption measurements.

References

[1] J. Casazza. *First the Tick, Now the Tock: Intel Microarchitecture (Nehalem). Intel® Xeon® processor 3500 and 5500 series Intel® Microarchitecture*. White Paper. Intel Cooperation, 2009.

[2] J. Charles et al. *Evaluation of the Intel® Core™ i7 Turbo Boost feature*. IEEE, 2009.

[3] N. R. Herbst et al. "BUNGEE: An Elasticity Benchmark for Self-Adaptive IaaS Cloud Environments". In: *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2015)*. Firenze, Italy: IEEE, 2015.

[4] B. Acun, P. Miller, and L. V. Kale. *Variation Among Processors Under Turbo Boost in HPC Systems*. ACM, 2016.

[5] K.-D. Lange and M. G. Tricker. *Server Efficiency Rating Tool (SERT) Design Document 2.0.3*. Tech. rep. Standard Performance Evaluation Corporation (SPEC), 2019.