Motivation and Requirements for a Many-core Approach

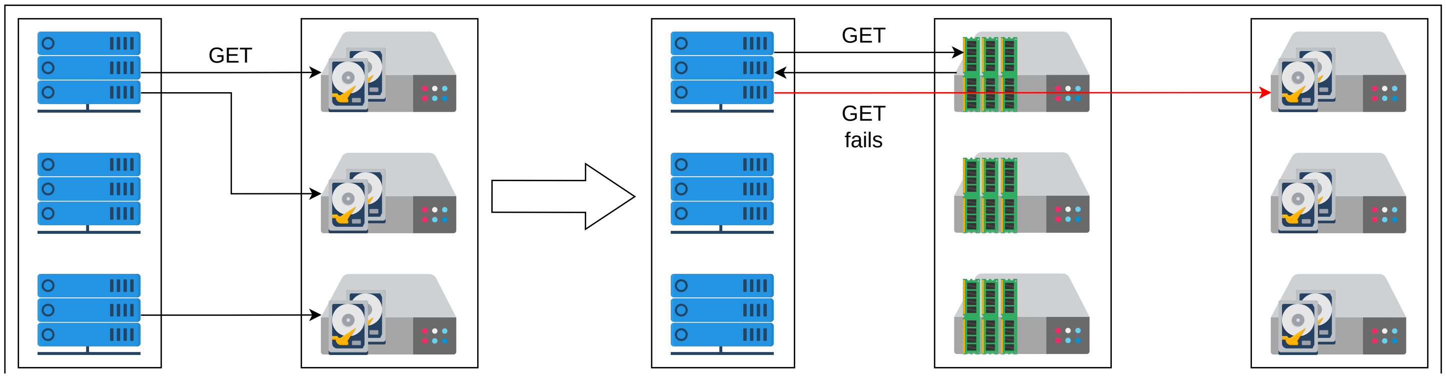
Learn about problems with scaling the Memcached key-value store with a many-core processor.

We'll cover the following

- Why we use key-value stores
- Problem statement
- Defining performance
- Factors affecting energy efficiency
  - Clock speed
  - Processor cores
  - Types of cores
- Problems with the caching software
- Requirements
  - Functional
  - Non-functional
- Bird's eye view

Why we use key-value stores

Memcached is a key-value store that is used by large platforms like Facebook and LinkedIn to read and write data quickly. Applications that require repeated fast reads and writes are ideal for a caching layer, which can be implemented using key-value stores.



Traditional architecture (left) vs. architecture with key-value store (right)

The image above shows how Memcached comes between the frontend and database to provide faster response time. Web architecture that uses a memory resident key-value store can be many folds faster than disk-based stores (typical RAM access takes 100 nanoseconds, while typical hard disk access takes four milliseconds). Key-value stores can be used to store:

- User preferences
- Users shopping carts
- Real time product recommendations

As the users of these platforms grow larger, the need to scale key-value stores has become essential, and one of the hurdles to scaling them is the financial cost, mostly electricity cost for running them. According to one study, the cost of powering up servers in a modern data center can be up to 50% of a typical three-year total cost of ownership (TCO).

Problem statement

To reduce the financial cost of data centers hosting key-value stores, we need to reduce their power consumption. To do that, we want to maximize energy efficiency by increasing the performance of our system while decreasing the power it consumes.

The Memcached key-value store has a `GET` operation that can retrieve multiple keys in a single call (often called a batched `GET`), which reduces overall network traffic. However, this batch operation requires relatively high RAM than the simple `GET` request because we would like to have a higher chance of finding all the keys on the server rather than ending up making a database call or accessing multiple key-value servers. So, the more RAM we can add to our Memcached servers, the better, but "wimpy" nodes support a lot less RAM than required. As a result, data centers have resorted to using power-hungry processors that can support more RAM, thereby increasing power consumption per node.

To maximize performance and minimize power consumption, we'll be using the performance per watt metric (instead of looking at just the performance or power usage alone):

$$\text{Energy efficiency} = \frac{\text{Performance}}{\text{Watt}}$$

Defining performance

Performance can be quantified using many metrics. We will use the following metrics to measure performance:

1. **Responsiveness (seconds):** The ability of a system to complete a task in a given amount of time. We are specifying it as the median response time to a request.
2. **Throughput (read/second or write/second):** The number of operations that can happen in a given second.

Factors affecting energy efficiency

The factors below contribute toward energy efficiency.

Clock speed

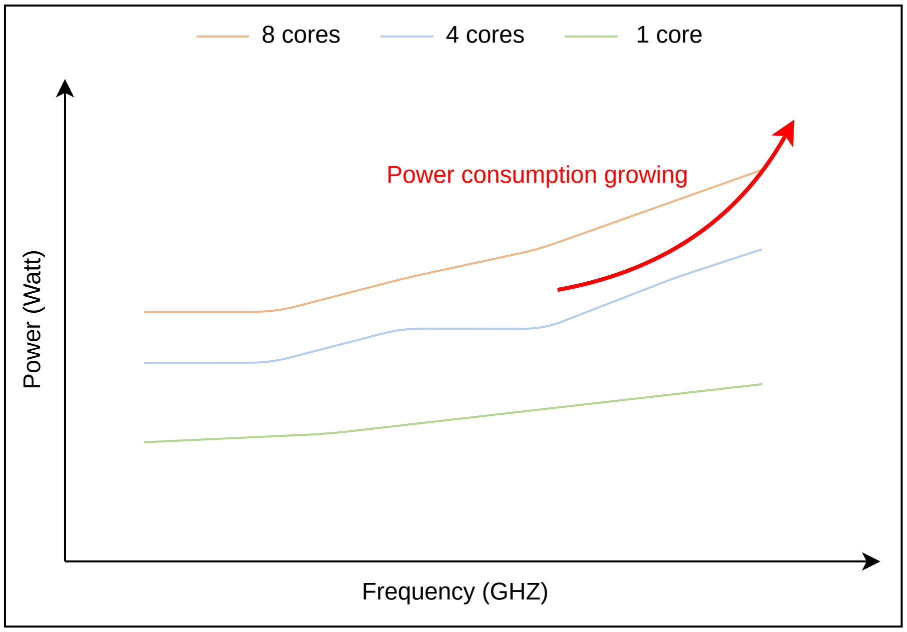
Clock speed refers to the number of cycles a processor executes per second. A higher clock speed is tempting because it translates to faster execution of instructions.

Per Moore's law, we continue to increase the transistor counts in our cores. Unfortunately, we still reach a ceiling in our processor clock speeds because power consumption is approximately directly proportional to the cube of the clock speed.

$$\begin{gathered}\text{Frequency} \approx \text{Voltage} \\ \text{PowerConsumption} \approx \text{Voltage}^2 \times \text{Frequency}\end{gathered}$$

$$\text{PowerConsumption} \propto \text{ClockSpeed}^3$$

This means that the power consumption will increase manifolds, making it harder to cool the processor from its surface when we try to increase the clock speed. The industry shifted to multi-core processors to get around this performance ceiling. However, multi-core processors require programmers to use multiple execution units to actually realize any higher performance.



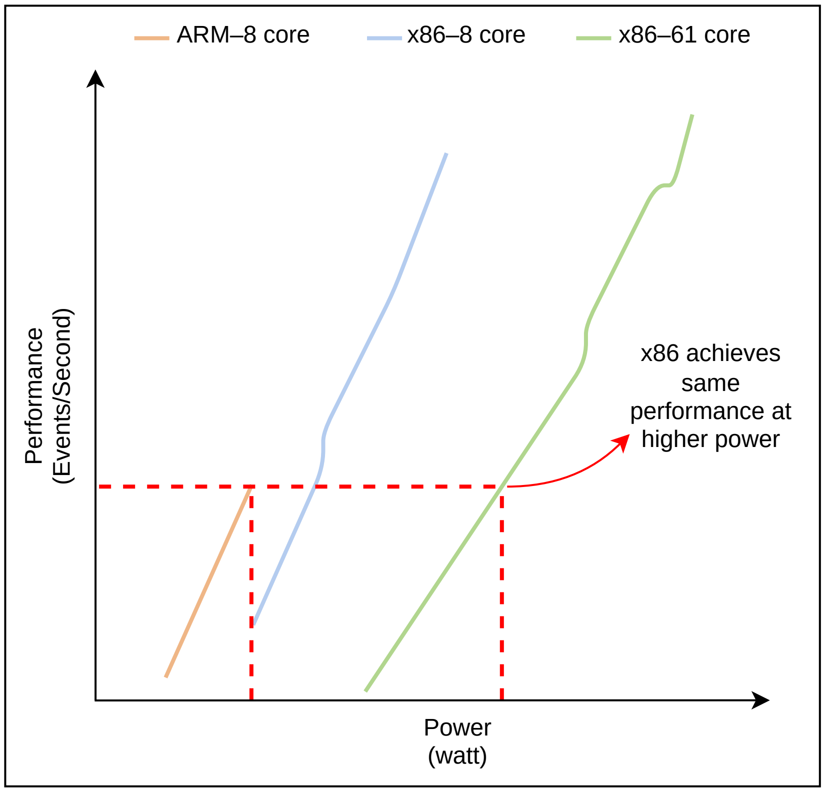
Relationship between the clock rate and CPU power

As you can see in the image above, the power consumption increases as the clock speed increases.

Processor cores

In a multi-core setup, Intel processors with x86 architectures have higher energy consumption when compared to a processor based on the ARM architecture.

As you can see from the image on the right, an ARM-based many-core processor can achieve better performance at lower power consumption. While an x86-based processor needs a lot more power to achieve the same performance, we want to maintain a lower power consumption with moderate performance even though the x86 processor has the capability to go above that performance.



Performance of APM XGene vs. Intel Xeon (2011)

Types of cores

1. A typical desktop class x86 processor is power-hungry but gives high performance. A typical desktop-class processor using this architecture uses around 130 watts.
2. ARM-based processors were initially designed for mobile form factors. While it was difficult for x86 architecture to pivot to mobile form factors, over the years, ARM has shown promising applications in broader use cases. Mobile ARM-based processors use around 5 watts.

3. The Tile architecture is a domain-specific architecture for servers from Tilera (currently this architecture design is owned by Nvidia after a series of acquisitions). General-purpose processors were designed to give as much performance as possible, and there was little focus on energy efficiency. This, however, has changed, and companies like Tilera are looking at domain-specific processors to provide power-efficient solutions for large-scale applications. Now, developers need to optimize software to take advantage of these performance improvements.

Problems with the caching software

Before the paradigm shift to multi-core processors, the software could be made faster by relying on the year-on-year increase in clock speeds of processors. However, to effectively utilize multi-core and domain-specific processors, we want software that can explicitly parallelize tasks (so that every core is busy doing work). Moreover, we want software that doesn't face performance bottlenecks from excessive locking in multi-threaded setups.

Requirements

We will be going over the requirements for our many-core key-value store below.

Functional

- **Data parallelization:** As we get more cores to work with, we need a key-value store that is more concurrent while accessing data.
- **Core–task mapping:** We'll want to find a close-to-optimal solution for allocating tasks to the multiple cores of the processor.

Non-functional

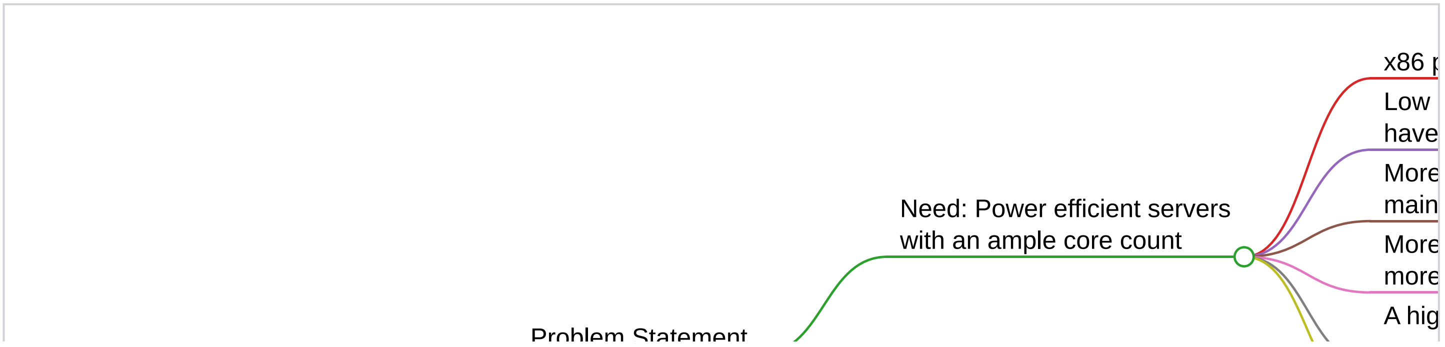
- **Low latency:** When using a power-efficient processor, we might get slower performance. However, the time our system takes to respond to a request should be less than one millisecond, so the user does not notice the loading. For many use cases, one millisecond is a reasonable latency.

- **High throughput:** Using parallelism, we'll increase the throughput, which is offset by the processor's slow performance.
- **Power efficiency:** We want to use less power to host more keys. We want fewer machines for easier management of data centers. Since we are using many more cores, we should be able to parallelize our tasks. The system should be able to respond to a higher number of requests at a given time while requiring less power. We need a system that comes in between the slow wimpy nodes and the nodes with great performance but power-hungry processors.

> In our design space, we have capital expense items (CAPEX) such as the dollar cost of a single server, the number and types of cores in a processor, total number of servers required for a use case, etc., as well as operational cost items (OPEX) such as power consumption, management cost of servers, etc. We want high throughput, low median latency, and good power efficiency. *As designers, we need to ask what we can change in our design space and how it will impact our required results.*

Bird's eye view

In the next few lessons, we will dive into the design of the many-core key-value stores. The following concept map is a quick summary of this chapter.


x86 p
Low
have
More
main
More
more
A hig
Need: Power efficient servers with an ample core count
Problem Statement