6.7 Clusters, warehouse scale computers, and other message-passing multiprocessors

The alternative approach to sharing an address space is for the processors to each have their own private physical address space. The figure below shows the classic organization of a multiprocessor with multiple private address spaces. This alternative multiprocessor must communicate via explicit *message passing*, which traditionally is the name of such style of computers. Provided the system has routines to *send* and *receive messages*, coordination is built in with message passing, since one processor knows when a message is sent, and the receiving processor knows when a message arrives. If the sender needs confirmation that the message has arrived, the receiving processor can then send an acknowledgment message back to the sender.

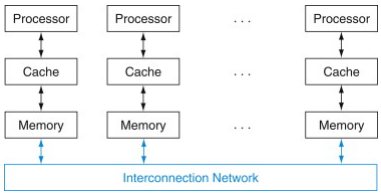
Message passing: Communicating between multiple processors by explicitly sending and receiving information.

Send message routine: A routine used by a processor in machines with private memories to pass a message to another processor.

Receive message routine: A routine used by a processor in machines with private memories to accept a message from another processor.

Figure 6.7.1: Classic organization of a multiprocessor with multiple private address spaces, traditionally called a message-passing multiprocessor (COD Figure 6.13).

Note that unlike the SMP in COD Figure 6.7 (Classic organization of a shared memory multiprocessor), the interconnection network is not between the caches and memory but is instead between processor-memory nodes.



There have been several attempts to build large-scale computers based on high-performance message-passing networks, and they do offer better absolute communication performance than clusters built using local area networks. Indeed, many supercomputers today use custom networks. The problem is that they are much more expensive than local area networks like Ethernet. Few applications today outside of high-performance computing can justify the higher communication performance, given the much higher costs.

Hardware/Software Interface

Computers that rely on message passing for communication rather than cache coherent shared memory are much easier for hardware designers to build (see COD Section 5.8 (A common framework for memory hierarchy)). There is an advantage for programmers as well, in that communication is explicit, which means there are fewer performance surprises than with the implicit communication in cache-coherent shared memory computers. The downside for programmers is that it's harder to port a sequential program to a message-passing computer, since every communication must be identified in advance or the program doesn't work. Cache-coherent shared memory allows the hardware to figure out what data need to be communicated, which makes porting easier. There are differences of opinion as to which is the shortest path to high performance, given the pros and cons of implicit communication, but there is no confusion in the marketplace today. Multicore microprocessors use shared physical memory and nodes of a cluster communicate with each other using message passing.

Some concurrent applications run well on parallel hardware, independent of whether it offers shared addresses or message passing. In particular, task-level parallelism and applications with little communication—like Web search, mail servers, and file servers—do not require shared addressing to run well. As a result, *clusters* have become the most widespread example today of the message-passing parallel computer. Given the separate memories, each node of a cluster runs a distinct copy of the operating system. In contrast, the cores inside a microprocessor are connected using a high-speed network inside the chip, and a multichip shared-memory system uses the memory interconnect for communication. The memory interconnect has higher bandwidth and lower latency, allowing much better communication performance for shared memory multiprocessors.

Clusters: Collections of computers connected via I/O over standard network switches to form a message-passing multiprocessor.

The weakness of separate memories for user memory from a parallel programming perspective turns into a strength in system dependability (see COD Section 5.5 (Dependable memory hierarchy)). Since a cluster consists of independent computers connected through a local area network, it is much easier to replace a computer without bringing down the system in a cluster than in a shared memory multiprocessor. Fundamentally, the shared address means that it is difficult to isolate a processor and replace it without heroic work by the operating system and in the physical design of the server. It is also easy for clusters to scale down gracefully when a server fails, thereby improving **dependability**. Since the cluster software is a layer that runs on top of the local operating systems running on each computer, it is much easier to disconnect and replace a broken computer.

Given that clusters are constructed from whole computers and independent, scalable networks, this isolation also makes it easier to expand the system without bringing down the application that runs on top of the cluster.

Their lower cost, higher availability, and rapid, incremental expandability make clusters attractive to service Internet providers, despite their poorer communication performance when compared to large-scale shared-memory multiprocessors. The search engines that hundreds of millions of us use every day depend upon this technology. Amazon, Facebook, Google, Microsoft, and others all have multiple datacenters each with clusters of tens of thousands of servers. Clearly, the use of multiple processors in Internet service companies has been hugely successful.

> "Anyone can build a fast CPU. The trick is to build a fast system.
> *Seymour Cray, considered the father of the supercomputer.*

Warehouse-scale computers

Internet services, such as those described above, necessitated the construction of new buildings to house, power, and cool 100,000 servers. Although they may be classified as just large clusters, their architecture and operation are more sophisticated. They act as one giant computer and cost on the order of $150M for the building, the electrical and cooling infrastructure, the servers, and the networking equipment that connects and houses 50,000 to 100,000 servers. We consider them a new class of computer, called *Warehouse-Scale Computers* (WSC).

Hardware/Software Interface

The most popular framework for batch processing in a WSC is MapReduce [Dean, 2008] and its open-source twin Hadoop. Inspired by the Lisp functions of the same name, Map first applies a programmer-supplied function to each logical input record. Map runs on thousands of servers to produce an intermediate result of key- value pairs. Reduce collects the output of those distributed tasks and collapses them using another programmer-defined function. With appropriate software support, both are highly parallel yet easy to understand and to use. Within 30 minutes, a novice programmer can run a MapReduce task on thousands of servers.

For example, one MapReduce program calculates the number of occurrences of every English word in a large collection of documents. Below is a simplified version of that program, which shows only the inner loop and assumes just one occurrence of all English words found in a document:

```
map(String key, String value):
   // key: document name
   // value: document contents
   for each word w in value:
      EmitIntermediate(w, "1");   // Produce list of all words
reduce(String key, Iterator values):
   // key: a word
   // values: a list of counts
   int result = 0;
   for each v in values:
      result += ParseInt(v);      // get integer from key-value pair
   Emit(AsString(result));
```

The function `EmitIntermediate` used in the Map function emits each word in the document and the value one. Then the Reduce function sums all the values per word for each document using `ParseInt()` to get the number of occurrences per word in all documents. The MapReduce runtime environment schedules map tasks and reduce tasks to the servers of a WSC.

At this extreme scale, which requires innovation in power distribution, cooling, monitoring, and operations, the WSC is a modern descendant of the 1970s supercomputers—making Seymour Cray the godfather of today's WSC architects. His extreme computers handled computations that could be done nowhere else, but were so expensive that only a few companies could afford them. This time the target is providing information technology for the world instead of high-performance computing for scientists and engineers. Hence, WSCs surely play a more important societal role today than Cray's supercomputers did in the past.

While they share some common goals with servers, WSCs have three major distinctions:

1. *Ample, easy parallelism:* A concern for a server architect is whether the applications in the targeted marketplace have enough parallelism to justify the amount of parallel hardware and whether the cost is too high for sufficient communication hardware to exploit this parallelism. A WSC architect has no such concern. First, batch applications like MapReduce benefit from the large number of independent data sets that need independent processing, such as billions of Web pages from a Web crawl. Second, interactive Internet service applications, also known as *Software as a Service (SaaS)*, can benefit from millions of independent users of interactive Internet services. Reads and writes are rarely dependent in SaaS, so SaaS rarely needs to synchronize. For example, search uses a read-only index and email is normally reading and writing independent information. We call this type of easy parallelism *Request-Level Parallelism*, as many independent efforts can proceed in parallel naturally with little need for communication or synchronization.

2. *Operational Costs Count:* Traditionally, server architects design their systems for peak performance within a cost budget and worry about energy only to make sure they don't exceed the cooling capacity of their enclosure. They usually ignored operational costs of a server, assuming that they pale in comparison to purchase costs. WSCs have longer lifetimes—the building and electrical and cooling infrastructure are often amortized over 10 or more years—so the operational costs add up: energy, power distribution, and cooling represent more than 30% of the costs of a WSC over 10 years.
3. *Scale and the Opportunities/Problems Associated with Scale:* To construct a single WSC, you must purchase 100,000 servers along with the supporting infrastructure, which means volume discounts. Hence, WSCs are so massive internally that you get economy of scale even if there are few WSCs. These economies of scale led to *cloud computing*, as the lower per unit costs of a WSC meant that cloud companies could rent servers at a profitable rate and still be below what it costs outsiders to do it themselves. The flip side of the economic opportunity of scale is the need to cope with the failure frequency of scale. Even if a server had a Mean Time To Failure of an amazing 25 years (200,000 hours), the WSC architect would need to design for five server failures every day. COD Section 5.16 (Fallacies and pitfalls) mentioned annualized disk failure rate (AFR) was measured at Google at 2% to 4%. If there were four disks per server and their annual failure rate was 2%, the WSC architect should expect to see one disk fail every *hour*. Thus, fault tolerance is even more important for the WSC architect than for the server architect.

Software as a service (***SaaS***) : Rather than selling software that is installed and run on customers' own computers, software is run at a remote site and made available over the Internet typically via a Web interface to customers. SaaS customers are charged based on use versus on ownership.

The economies of scale uncovered by WSC have realized the long dreamed of goal of computing as a utility. Cloud computing means anyone anywhere with good ideas, a business model, and a credit card can tap thousands of servers to deliver their vision almost instantly around the world. Of course, there are important obstacles that could limit the growth of cloud computing—such as security, privacy, standards, and the rate of growth of Internet bandwidth—but we foresee them being addressed so that WSCs and cloud computing can flourish.

To put the growth rate of cloud computing into perspective, in 2012 Amazon Web Services announced that it adds enough new server capacity *every day* to support all of Amazon's global infrastructure as of 2003, when Amazon was a $5.2Bn annual revenue enterprise with 6000 employees.

Now that we understand the importance of message-passing multiprocessors, especially for cloud computing, we next cover ways to connect the nodes of a WSC together. Thanks to **Moore's Law** and the increasing number of cores per chip, we now need networks inside a chip as well, so these topologies are important in the small as well as in the large.

Elaboration

The MapReduce framework shuffles and sorts the key-value pairs at the end of the Map phase to produce groups that all share the same key. These groups are next passed to the Reduce phase.

Elaboration

Another form of large-scale computing is grid computing, where the computers are spread across large areas, and then the programs that run across them must communicate via long haul networks. The most popular and unique form of grid computing was pioneered by the SETI@home project. As millions of PCs are idle at any one time doing nothing useful, they could be harvested and put to good use if someone developed software that could run on those computers and then gave each PC an independent piece of the problem to work on. The first example was the Search for ExtraTerrestrial Intelligence (SETI), which was launched at UC Berkeley in 1999. Over 5 million computer users in more than 200 countries have signed up for SETI@home, with more than 50% outside the US. By the end of 2011, the average performance of the SETI@home grid was 3.5 PetaFLOPS.

Check yourself

1. True or false: Like SMPs, message-passing computers rely on locks for synchronization.
2. True or false: Clusters have separate memories and thus need many copies of the operating system.

Answer: 1. False. Sending and receiving a message is an implicit synchronization, as well as a way to share data. 2. True.