

Distributed Wisdom: Analyzing Distributed-System Performance—Latency vs. Throughput

Roy Friedman, *The Technion*

Erez Hadad, *The Technion*

Many textbooks and articles have discussed the fact that latency and throughput aren't opposites. Consider the well-known comparison of the throughput of a modern cargo ship packed with tapes on a two-week journey with the bandwidth in today's fastest networks. The cargo ship wins big-time. Clearly, if you wish to send a small packet, the Internet is a better option. However, for transferring a very large database, low-tech options would prove faster.

Technical people often forget this obvious observation. For example, in the summer of 1997, one of us attended a talk by a founder and the CTO of a leading search engine. He explained that his company's Web site was hosted in California with a mirror on the East Coast. His search engine updated its content twice a week, because the company wants to keep the main site and its mirror synchronized, and copying the entire database over the Internet took 72 hours. An attendee asked, "In this case, why don't you store it on a tape and send it with overnight delivery?" The speaker paused for a few seconds and replied, "Hmm, that's a good point. We never thought about it."

You might be asking yourself what this anecdote has to do with modern distributed systems. The answer is that the distributed-systems research community still often ignores the fact that as long as the latency is reasonable, throughput is really what matters. In particular, research papers often break down the latency costs, but how often do we see a breakdown of the throughput-limiting factors? In particular, the parts of the latency that are purely CPU-bound affect throughput, yet other parts that involve networking activity and waiting in buffers and queues often have a much smaller impact on throughput. This is true because modern software typically is multithreaded, so when

some activity is blocked on I/O or in a queue, the CPU can handle other client requests. Suppose that you get a latency breakdown indicating that some CPU-bound activity takes 30 percent of the latency and that another I/O-bound activity takes the other 70 percent. The usual instinct (as is evident in too many research papers) is to assume that because 70 percent of the latency is I/O-bound, there's no point in further optimizing the code. BIG MISTAKE!

Experiences with throughput improvement

Our motivation for writing this article came when we were working on FTS, a replication-based fault-tolerant service for CORBA objects.¹ In FTS, we replicate CORBA objects transparently and portably by using, among other techniques, dynamic skeletons and dynamic invocations² to catch client requests and multicast them to all other replicas, using a group communication toolkit. This way, the replication is portable and interoperable across different CORBA -compliant ORB (object request broker) implementations.

Because we work inside a dynamic-invocation mechanism, our code must access client requests and replies as standard pseudo-CORBA objects. So, to multicast them, we must serialize and deserialize them. The CORBA standard offers a portable way to do this. However, we discovered that this mechanism is highly inefficient, at least in the ORB that we used, which is a leading industry-supported open source ORB. We managed to implement our own mechanism, which is still portable yet 50 percent more efficient than the other mechanism. Even though this portable serialization and deserialization cost at most 10 percent of the total latency (and often much less than that), the maximal throughput improved by approximately a factor of two. In other words, the throughput improvement was roughly equivalent to the improvement that we made to the portable serialization and deserialization alone!³

To illustrate this, figure 1 shows a qualitative breakdown of the latency for serving a request. S_1 denotes the cost of serialization and deserialization in the standard implementation; S_2 denotes this cost in the optimized implementation. I denotes the rest of the latency. The throughput improvement that we measured was proportional to the ratio between S_1 and S_2 , rather than to the ratio between the overall latencies—that is, between $I + S_1$ and $I + S_2$. This is because most of the rest of the latency was indeed largely caused by the network and due to waiting in queues and buffers.

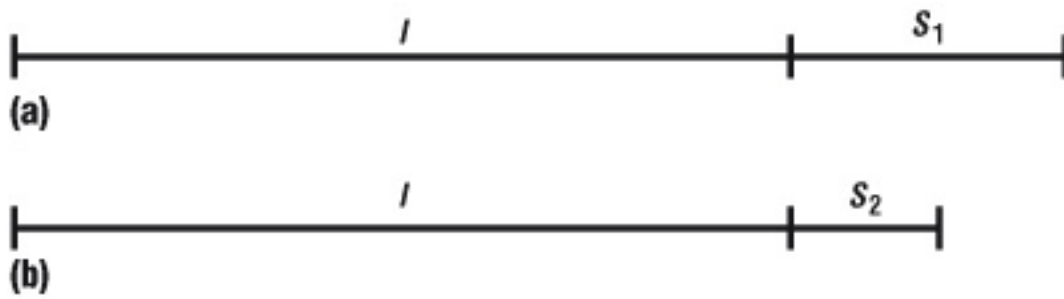


Figure 1. The qualitative latency breakdown for serving a request for (a) standard and (b) optimized implementations. S_1 denotes the cost of serialization and deserialization in the standard implementation; S_2 denotes this cost in the optimized implementation. I denotes the rest of the latency.

Interestingly, when fixing the offered load on the system, you can observe an even more radical phenomenon. Under a heavy load, the optimizations we've discussed improved the systems' latency and throughput by a much greater factor. Moreover, in these highly loaded situations, the serialization and deserialization costs decreased to less than 0.1 percent of the overall latency. So, how can this be?

If we take a more refined view of how a request is serviced, we can identify three types of activities (see figure 2): CPU bound activities, denoted by C ; I/O operations, denoted by I ; and the time W spent waiting for the CPU when it's busy with other threads. Assume we can model the service time at the CPU and the arrival of new requests as Poisson processes. Then, queuing theory tells us that W grows to infinity as the interarrival rate approaches the service rate (see figure 3), which actually happens during a high load. In these situations, any small change in C implies a huge difference for W .



Figure 2. More accurate latency breakdown: I denotes I/O operations, W denotes waiting for the CPU, and C represents computational time at the CPU.

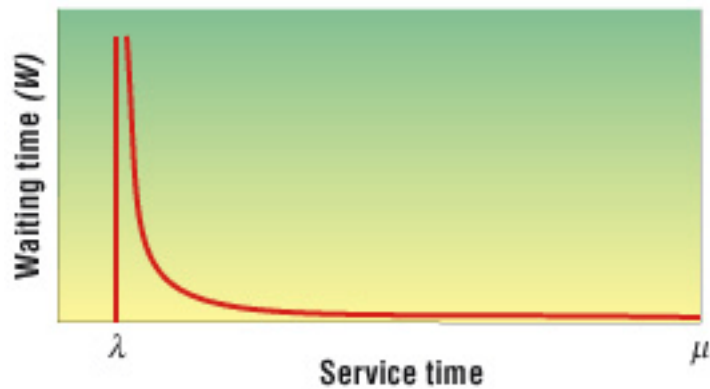


Figure 3. The waiting time (queue length) versus the service time at the CPU μ for a given offered load whose interarrival rate is λ .

Notes on serialization and deserialization

Portable serialization and deserialization are very important, for example, in realizing Fault-Tolerant CORBA's replication demands, as well as for several applications beyond replication. One example is a logging service needing to serialize data while a corresponding recovery service needs to deserialize it. Similarly, a persistent-state service also must serialize and deserialize data. To that end, the CORBA standard would benefit from adding a set of primitives for direct and efficient serialization and deserialization of CORBA entities to and from byte sequences.

Conclusion

So, in analyzing distributed-system performance, it's not enough to include a latency breakdown (even though it's the easiest performance measure to get). Instead, you should indicate next to each item whether the activity is CPU-bound or I/O-bound. Quite often, CPU-bound activities take a small portion of the overall latency but have a much bigger impact—on throughput and indirectly on the overall latency.

References

1. R. Friedman and E. Hadad , "FTS: A High-Performance CORBA Fault-Tolerance Service," *Proc. 7th IEEE Int'l Workshop Object-Oriented Real-Time Dependable Systems* (Words 02), IEEE CS Press, 2002, pp. 61-68.
2. *CORBA /IIOP Specification 2.4.2*, <http://www.omg.org/cgi-bin/doc?formal/01-02-33>, document formal/2001-02-33, Object Management Group, 2001.
3. R. Friedman and E. Hadad , *A Case for Efficient Portable Serialization in CORBA* , tech. report, Dept. of Computer Science, Technion, 2005.



Roy Friedman is a senior lecturer in the Technion's Computer Science Department. Contact him at roy@cs.technion.ac.il.



Erez Hadad is a PhD student at the Technion's Computer Science Department. Contact him at erezh@cs.technion.ac.il.

Related Links

DS Online's Dependable Systems Community, <http://dsonline.computer.org/dependable>

"Simulating Commercial Java Throughput Workloads: A Case Study", <http://doi.ieeecomputersociety.org/10.1109/ICCD.2005.97>

"Zone-Based Traffic Load Balancing to Ensure Quality of Service (QoS) in Distributed Systems", <http://doi.ieeecomputersociety.org/10.1109/ICPADS.2005.300>

"System Throughput Maximization Subject to Delay and Time Fairness Constraints in 802.11 WLANs", <http://doi.ieeecomputersociety.org/10.1109/ICPADS.2005.273>

Cite this article: Roy Friedman and Erez Hadad, "Analyzing Distributed-System Performance: Latency vs. Throughput," *IEEE Distributed Systems Online*, vol. 7, no. 1, 2006, art. no. 0601-o1001.