Inside Spatial Server Hardware

Jonathan W. Lowe

Demystifying the hardware components that comprise geo-enabled Web servers — and learning how to own one yourself — isn't as daunting as you might think.

How many geospatial practitioners do you know who work at computer terminals every day but would say of themselves "I'm not really much of a hardware person"? Given hardware's confusing acronyms, intimidating complexity, and fast-paced evolution, it's easy to understand why so many of us take that stance. There are two increasingly good reasons, however, for developers of interactive mapping Web sites to become more confident about hardware. These include the decreasing costs of both servers and colocation. In this column, I'll consider the benefits of owning and colocating a Web server, then I'll describe how each hardware component of a server contributes to a processing task. With these relationships in mind, we can all think more like "hardware people" when configuring or upgrading our hardware for optimal spatial application performance.

Carrots and Sticks

Currently, the best reason to administer your own online presence is the ever-lower price of hardware and ISP (Internet service provider) colocation fees. Computers powerful enough to serve interactive browser-based maps to an entire municipality now cost as little as $500. And colocating your new server at an ISP's facility costs $300 or less per month. So, for a $50 initial investment, a $30 monthly charge, some free open-source GIS software, and copious spare time for skills development, ten starving GIS grad students can share a Linux box and host their ten different interactive mapping projects online. And if trends continue, these already minimal costs will continue to drop as hardware, speed, capacity, and power increase.

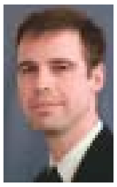
Of course, there are other ways to get an interactive Web-mapping site online without buying your own hardware and colocating it. In addition to offering colocation, some companies (such as Metropolis New Media in San Jose, California), will configure a server with GIS software then rent subdivisions of its space to multiple customers. The problem is that renting space on someone else's hardware means you no longer have full control of your data and application code. Other users sharing the space on the same server may tie up its resources in unexpected ways. Still worse, someone else — the 800-pound gorilla otherwise known as the system administrator — can lock you out, browse through your directories, copy your data, and do anything you can do . . . and more. Data simply isn't secure if you don't control the box that houses it.

Inside a Spatial Web Server

Determined to take control of my applications and data, I asked colleagues for their spatial-specific hardware recommendations. They confirmed that spatial processing requires special hardware configuration. For instance, spatial database expert and open-source advocate Paul Ramsey of Refractions Research, Inc., shared the following general hardware rules for a spatial Web server:

> GIS mapping is a data-reading exercise firstly and a rendering exercise secondly. Fast disks and a fast front-side bus can help you get data off of the media and onto your map very fast. RAID (Redundant Array of Independent Disks) 10 on fast-spinning disks is the fastest redundant option. Extra memory helps your operating system cache frequently used data off of the media. Fast CPU will speed your render.

There's a universe of knowledge between Ramsey's concise overview and the details of each hardware component and their interrelationships (see Figure 1). What are the key components of a typical server, and how do they work together? To better understand Ramsey's guidelines, we'll consider a day in the life of a spatial Web server at the internal hardware components level. This imaginary server's job is to provide interactive municipal maps of assessor's parcels and underlying aerial imagery to the general (Internet) public.



Net Results columnist **Jonathan W. Lowe** covers the role of emerging technologies in the exchange of spatial information. Lowe is the owner of Local Knowledge Consulting (Berkeley, California), where he designs and implements spatial Web sites. He can be contacted at info@giswebsite.com.

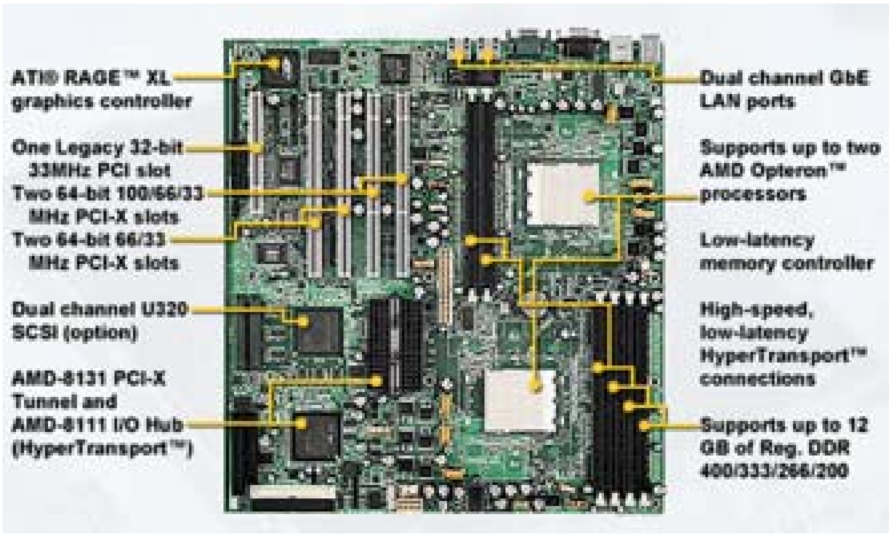

ATI® RAGE™ XL graphics controller
One Legacy 32-bit 33MHz PCI slot
Two 64-bit 100/66/33 MHz PCI-X slots
Two 64-bit 66/33 MHz PCI-X slots
Dual channel U320 SCSI (option)
AMD-8131 PCI-X Tunnel and AMD-8111 I/O Hub (HyperTransport™)
Dual channel GbE LAN ports
Supports up to two AMD Opteron™ processors
Low-latency memory controller
High-speed, low-latency HyperTransport™ connections
Supports up to 12 GB of Reg. DDR 400/333/266/200

Figure 1. A motherboard is to a computer what a chassis is to a car — the integration area for all subcomponents and their connections. This image of a Tyan Thunder K8S (S2880) motherboard annotates the locations, names, and (sometimes) throughputs of key hardware elements, but doesn't reveal their relationships to each other. Specification sheets give more detail about each part, but also count on the reader to know how the parts work together.

For starters, imagine an anonymous public user scrutinizing a mapping Web page displayed on his local browser. The page consists simply of a JPEG map image and navigation tools. Our curious user selects a zoom-in navigation tool and mouse-clicks on the map image. Whoosh! The user's browser sends his (form-driven) zoom request to our spatial Web server as an HTTP (hypertext transfer protocol) "get" request, initiating our day in the life.

Buses and Ports. On the receiving end, the Internet map-zoom request appears at our spatial Web server's Ethernet port (item A in Figure 2). The term port is hardware jargon for a shared linear pathway connecting two devices with a bidirectional communication channel. The more general term for such a connection (implying three or more devices) is bus.

Chipset. So, what devices does the Ethernet port connect? On one side of our server's Ethernet port is the vast territory of the Internet. On the other side, waiting to catch incoming data, is a device called the chipset (item B in Figure 2). If our server were a factory, the chipset would be its foreman, providing intelligence, coordination, and operational oversight to key hardware elements. For instance, the chipset arbitrates data transfers between the processor, memory, and other components — all of which may be operating at widely different speeds. Through clever direction, the chipset reduces wait times between slower and faster components.

Processor. Upon reception, the chipset passes the map-zoom request to the server's processor, also known as the central processing unit, or CPU (item C in Figure 2). The processor is the piece of hardware that performs calculations to produce a result in answer to a demand. The map-zoom request contains both a complicated demand ("Draw a map of extent X and size Y, containing layers A and B") and an implicit or explicit set of rules to follow when responding to that demand ("Determine the drawing steps using Web-server software such as Apache.") Eager to start work, the processor asks the chipset for access to those rules so it can begin evaluating the demand.

Storage Media. Rules are essentially lists of instructions (for example, programs) that are stored in files. When the processor requests instructions, the chipset has three likely places to find them, each a different storage medium: very fast cache, moderately fast memory (RAM, item D in Figure 2), or relatively slow hard disk

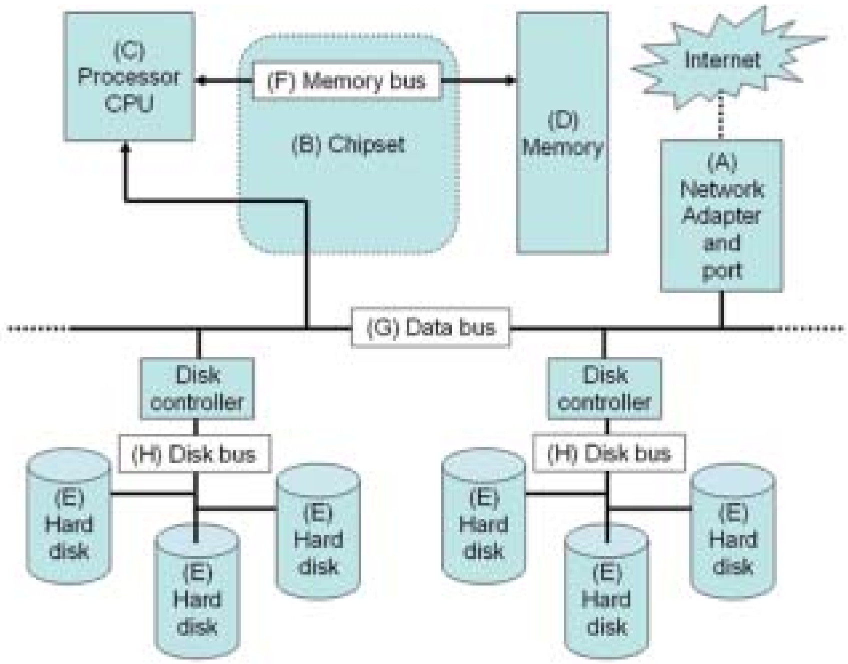


Figure 2. This logical data and workflow diagram shows key hardware components and their connections: network adapter/port, chipset, processor, cache and memory, hard drives, memory bus, data bus, and disk bus.

Facing the 800-Pound Gorilla

Does the idea of a system administrator taking over your Web site sound like a paranoid delusion? Consider this cautionary tale: In the May 2003 Net Results column, I reviewed a spatial hosting company called pgHoster and noted that such a shockingly inexpensive spatial Web-hosting option looked almost too good to be true. Heeding my own advice, I opened an account with pgHoster. Unfortunately, later in my experience as one of their customers, pgHoster's offering did turn out to be too good to be true. Access to my domain was unreliable, especially on weekends, and my support requests often went unanswered. For the low rate of $10 per month, though, temporary loss of service wasn't bad enough to turn me away — I designed my research to work around a maximum of three days without data collection.

Then in July 2004, following an attack by a cracker, the entire pgHoster site (and all domains it hosted) was compromised and taken completely offline for more than three consecutive weeks, ruining my research project. After partially recovering, pgHoster offered to restart the GIS databases if customers were willing to pay 150 percent more than the originally established rate. Meanwhile, I could not access my data or alter any files in my account. If not for my own backups, I would have had no room to negotiate.

Admittedly, administering an online spatial database hosting service is no small challenge, and pgHoster's staff likely meant well. But the bumps were finally too rough for me. To any readers who also selected pgHoster based on my then-positive review, please accept my humble apologies. The pgHoster idea remains as valuable as ever, but practical execution of that idea remains incomplete.

Combined with the low price of hardware and colocation, plus the scarcity of online spatial database hosts, this experience was my turning point to acquire and colocate the entire package myself.

(item E in Figure 2). The processor can accept and execute instructions very quickly. Consequently, the ideal storage medium supports equally fast reads and writes. Unfortunately, the fastest storage media are also the most expensive, so our spatial Web server has a very small amount of expensive cache (L1 and L2), a larger amount of less expensive memory (main memory), and finally, the affordable but slow hard-disk storage.

In our story, the processor's request for the Apache Web server's instruction set becomes one of many conflicting requests that the chipset must juggle with maximum speed and efficiency. The chipset must find and copy data-of-the-moment between slow and fast storage media. In practice, this means that the chipset first searches the cache for Apache instructions. If they're not already there, the chipset searches main memory, and then it searches the hard disks. As soon as it finds the instructions, the chipset displaces whatever was in the cache and puts some subset of the Apache instructions there instead, putting any remaining instructions into main memory. Once the instructions are in fast memory, they remain there and can be used repeatedly until displaced when some other program runs, or when the server is shut down.

With the instructions it needs in main memory, the processor can begin executing them consecutively. To transfer chunks of instructions between itself and memory, the processor uses a bus called the *memory bus*, (item F in Figure 2), or *frontside bus*, a three-way connection between processor, chipset, and memory with its own maximum speed and throughput.

Eventually, the processor discovers that Apache's is not the only instruction set that satisfies the map-zoom request. Still needed are vector and raster data from a database (Ramsey's “data-reading exercise”), as well as a program that renders these data as a graphic map (Ramsey's “rendering exercise”). As before, the chipset seeks out and transfers these additional programs' instructions to memory, and the processor crunches onward.

Some of the instructions tell the processor to request data from the local spatial database and copy it to main memory. In this case, data flow between the hard disks and main memory along both a data bus (item G in Figure 2) and a disk bus (item H in Figure 2). When the processor finishes consolidating these data in main memory, it requests access to another program to render those data as a graphic map. Ideally, the rendering program instructs the processor to draw the map directly to fast main memory — or even to cache — rather than to a file on the (slow) hard drive. Once again, the processor and memory exchange data across the frontside bus to produce a graphic map image.

After the rendering is complete, Apache resumes control, sending the new image and surrounding navigation tools from main memory back to the chipset, then to the Ethernet port, and, finally, back to the browser of the public user. The sun sets on our day in the life of a spatial Web server.

Balance and Bottlenecks

Practitioners who do consider themselves hardware people have similar processing flows in mind when building or upgrading spatial computers. Starting with an understanding of their spatial processing needs, they then step through a day in the life such as we did, noting the throughput of each hardware component (see Figure 3). They confirm that overall throughput on all devices and their connecting buses are as balanced as possible. One slow component in an otherwise fast mix will create a performance bottleneck. For instance, because every request must eventually pass between the processor and main memory to be executed, relatively slow memory bus throughput can limit performance of almost every task the system attempts. Thus, hardware people typically begin by checking frontside bus speed when conducting throughput analysis.

Furthermore, to keep us all on our toes, the potential bottleneck areas change as different components leapfrog forward in performance, turning previous leaders into laggards. For instance, today's Ethernet ports typically transfer data at a speed of 250 MB per second, but they are already making way for Ethernet replacements capable of moving 2.5 GB per second. Frontside-bus speeds used to hit a throughput ceiling of 1 GB per second. But with a new *hypertransport* technology, they can theoretically exceed 4 GB per second.

Revisiting Ramsey

This column's purpose was to demystify the relationships between the key hardware components of a spatial Web server to better understand where bottlenecks occur and how to mitigate them. Did the column succeed? With hardware balance and our example in mind, let's re-examine some of Ramsey's spatial hardware rules.

■ *Fast disks and a fast frontside bus can help you get data off of the media and onto your map very fast* — This makes sense. The flow from raw data to rendered map begins with the hard disk, moves to main memory via the disk and data buses, and then relies on the processor to quickly (via its frontside bus connection to main memory) grab and render that data. Different hard disks spin at different speeds, the fastest being the most expensive to purchase. The faster a disk spins, the less time it takes the cylinder to revolve to where the actuator arm and its head are over the sector of the disk that contains the data that the processor wants. As for the frontside bus, even if fast disks speed data transfer between disk and main memory, there will be a systemwide delay if exchanges between main memory and the processor are slow.

Bringing disk throughput (such as with the RAID approach, described below) almost as high as frontside bus throughput will maximize this data-to-map flow.

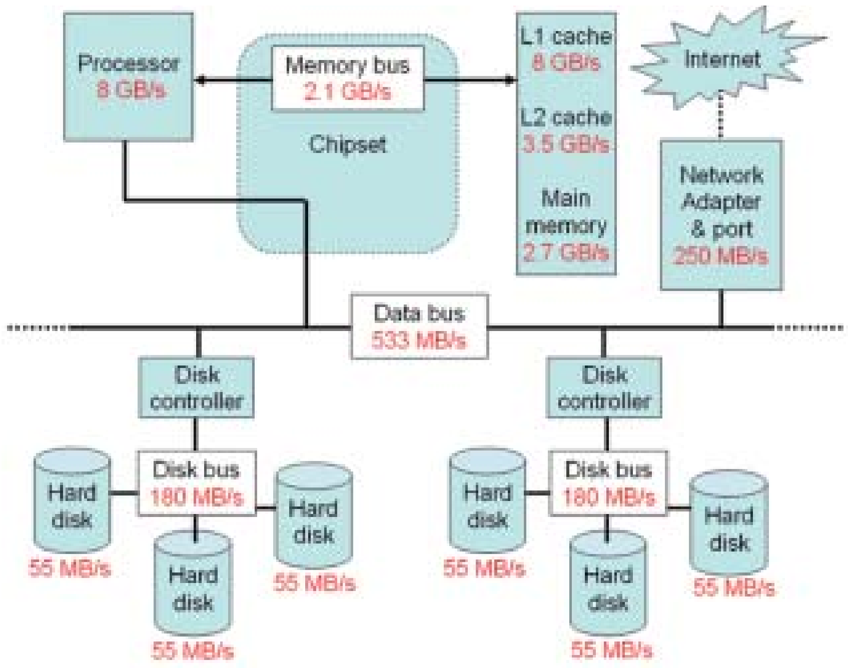


Figure 3. A logical data and workflow diagram with maximum throughput values (data per second) for each component and bus reveals the bottlenecks, if any, between them. Imagine flowing water in a pipe network instead of electronic data in circuitry. Where in this system would the water pool because of bottlenecks?

■ *RAID 10 on fast-spinning disks is the fastest redundant option* — In almost all cases, the slowest component of the mix will be the hard disks. Today's fastest-spinning disks (15,000 RPM) support throughput between 55 and 85 MB per second, far slower than the typical processor, main memory, or frontside bus. Spreading the data across a RAID, however, allows the chipset to access all disks at the same time, bringing the total throughput of an array of 10 55 MB-per-second disks to 550 MB per second.

There are several popular RAID strategies (RAID 0, 1, 5, and 10) for distributing data across these disk arrays, such that no data is lost even if one or more of the disks crash and overall disk throughput is balanced in comparison with other devices in the system.

■ *Extra memory helps your operating system cache frequently used data off of the media* — A spatial Web server may repeat the same kind of actions, such as rendering maps, many times in a row before getting any other demands.

When this happens, the instructions and data stored in cache and main memory from the previous demand are already in place for fast access on consequent similar processing. The larger the main memory and cache, the more of these repeatable instructions or reusable data are directly accessible without a slow trip to the hard disk and back.

Hands On

During my junior and senior years as an undergraduate geologist, I had to unlearn and relearn everything I had picked up as a freshman and sophomore. The simplifications needed to initially grasp concepts supplied the context for discovery of more subtle details and shades of gray later on. Having a general mental map of the inner workings of your computer is the first step. Configuring, purchasing, and experimenting with real hardware is the next and will doubtless change your original frame of reference. Best of luck getting the most performance for your investment and learning from direct experience! ⊕