www.fotolia.de, andyb1126

Tips for optimizing Apache, Postfix, Oracle, MySQL, and Samba

READY FOR TRAFFIC

Your homepage was just linked by Slashdot, a new email campaign goes out tonight, and you need the database to deliver survey results. We'll show you how to help your servers survive the strain. **BY BADRAN F. ARWATI, PEER HEINLEIN, RALF HILDEBRANDT, CHARLY KÜHNAST, AND VOLKER LENDECKE**

Server optimization is a question of survival: a server's efficiency drops when the load increases. And if incoming requests push the load too high, the system could crash. To avoid bringing down critical services, you need to be proactive. Faster hardware might not be an option, but a better configuration and some simple optimization steps might help. In this article, five experts provide an inside look at how to maximize throughput on your web, mail, database, and file servers.

A Healthy Start

No matter what kind of servers you manage, there are a few basic rules that every system administrator should know:

1. Pay attention to the mix! One factor often determines the overall system performance: CPU, I/O, or memory usage. It makes sense to distribute services to avoid running two services with the same performance-defining factor on the same server (Figure 1).

For example a mail server's speed is typically restricted by disk and network latency. Of course, the integrated virus scanner will keep the CPU busy on some servers, but if not, the CPU on a server that does nothing but distribute mail should have plenty of time to handle other tasks.

2. Monitor everything! The only way of discovering the key performance factors is to monitor operations over an extended period; this will give you values for the CPU, I/O, or memory usage during normal operations. *vmstat* and *sar* disk functions analyze disk throughput; *top*, *htop*, *uptime*, or *sar* can help you monitor the CPU; *ps*, *top*, or *sar* can help you track memory consumption.

You can extrapolate a full load scenario from the values for normal load. SNMP based monitoring (using Nagios, for example) will warn you of imminent disaster. After identifying the bottlenecks, don't forget to disable unneeded logging.

3. Excessive swapping is every system administrator's nightmare. Many back-end services let you configure an upper threshold for the maximum number of active instances. As swapping memory in and out drastically impacts other disk access, setting these thresholds too high will leave more and more processes waiting for the system to handle their I/O requests, which in turn leads to even more swapping – a vicious circle.

All the services taken together should not be allowed to spawn more instances than the server machine's physical memory can hold.

RAM or Disk?

4. Never waste your server's memory. Many back-end services can be streamlined; make sure you only load the modules you need, or rebuild packages to match your needs. This gives you more space for instances in RAM. By default, Apache will typically load a bunch of unnecessary services; if you have Postfix, you might consider building binaries without MySQL, TLS, or LDAP support. The memory footprint of CDB is much smaller than that of the Berkeley DB. With read-only maps, changing the map type will often save memory.

5. Clearly separate partitions for data, system, and logs! Separate partitions give you the ability to select the best filesystem for the job (for example, Ext 3 for the system partition, and XFS for the data partition). Some disk or RAID systems avoid hard disk thrashing in case of competing, concurrent access.

6. Log the actions your services perform! Without logs, you will not have any data to analyze or troubleshoot. A - prefix in the logfile name in */etc/syslog.conf* enables asynchronous writing and reduces the load on the filesystem.

Keep Safely

7. Back up your current configuration before you make a change. Small changes might affect your server's performance in a completely unprecedented way, and finding out why can take up a lot of your valuable time. A system administrator should thus use version control for configuration files, or at least create a backup before implementing changes. This gives you the ability to react if customers complain of sudden performance hits.

8. Store everything that doesn't change in a cache! Caches can be a big help in many situations: reverse proxies (such as Squid) upstream of database-based CMS systems can reduce the load on the database. Caching DNS servers (such as Dns-Cache, or Bind) remove the need for log analyzers and mail servers to perfom DNS lookups. The internal cache in the Amavisd New virus scanner prevents repeated analysis of the same content.

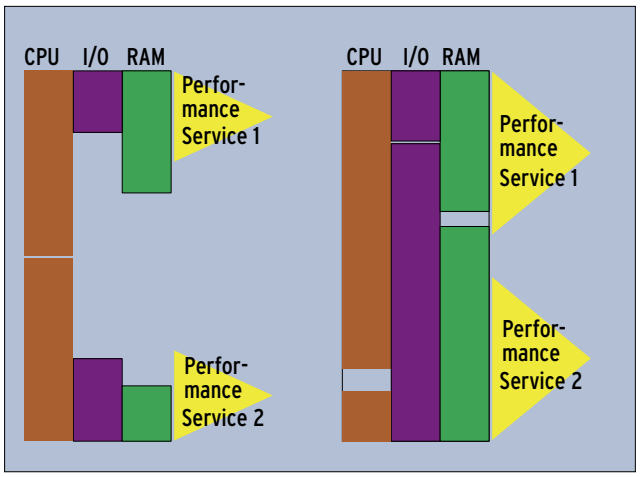

Figure 1: Two services with completely different CPU cycle, memory, and I/O load requirements leverage the power of a server far better than applications that compete for resources.

Use a Doorkeeper

9. Get rid of uninvited guests as early as possible! Users without access to the server can't create load on the server. A firewall, access controls, and *smtpd_*_checks* in Postfix send uninvited guests packing before they have a chance to generated unwanted system load. The Anvil server in Postfix [1] will additionally restrict the number of messages the server accepts over a unit of time to a level that will prevent performance loss due to queueing. In a similar way, Cband [2] supports bandwidth limits on the Apache web server.

10. Knock softly! Port knocking is a resource-saving way of keeping the firewall completely tight but still allowing trusted users to log on. Measures such as using one-time passwords, or relocating services such as SSH to unknown ports, are good for security, and they'll prevent uninvited guests from stressing your CPU.

▶ Web Servers

If your URL is published on a high-profile site, you can expect a dramatic increase in visitors. The following steps will help your servers handle the load.

1. Take care to select the right multiprocessing module! The *prefork* MPM forks a number of identical Apache processes and is best suited to machines with up to two CPUs. The more CPUs your web server has, the more likely it is that the *worker* MPM, which uses multiple threads per process, is the better choice.

2. Make good use of the cache! Apache has two mechanisms, *mod_disk_cache* and *mod_mem_cache*, for caching frequently requested content. If you have a lot of RAM (and, after all, there is no replacement for RAM), *mod_mem_cache* [3] is your best option.

Ditch the Ballast

3. Ditch your ballast! The Htaccess mechanism may be useful, but it is also a performance killer. So, get rid of it if you don't need it. *AllowOverride None* will remove the need for time-consuming parsing of *.htaccess*.

4. Ditch even more ballast! Sysadmins will also want to remove symlinks (*Options -FollowSymLinks*) and any modules they don't need. The perfect solution is to build a static version of Apache with everything you need, and not to load any modules at all at runtime.

5. Do without lookups! Hostname lookups will slow down even the fastest nameserver. *HostnameLookups off* removes the bottleneck. If you really need this information, you can perform any lookups you need later when you review your logs with a tool such as Webalizer.

6. Honor your clients, and don't make them wait. The *MaxClients* directive is critical for web performance. If you set too low a value for *MaxClients*, not all clients will be serviced in a timely fashion; if the value is too high, your clients will be forced to wait in the TCP queue. The only way to discover the right value is load testing.

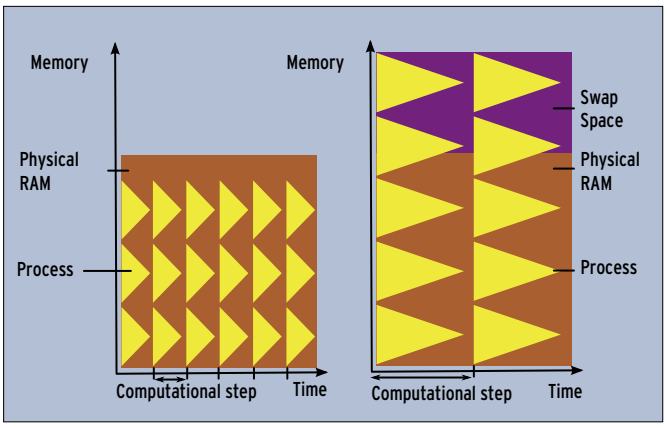

Figure 2: Classic crash scenario: if the number of processes under load increases so drastically that the machine starts to swap, the drop in throughput will increase the load even more.

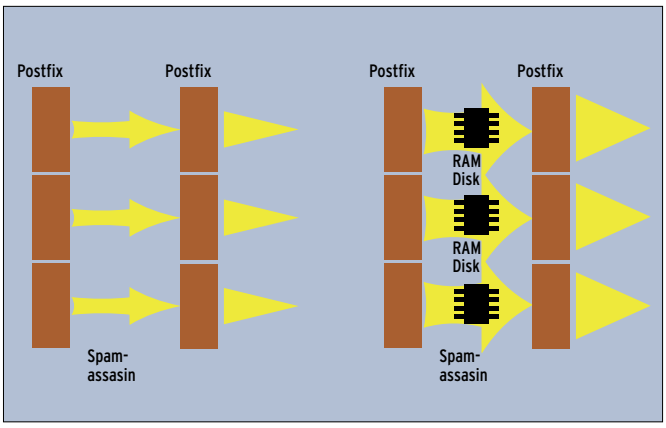

Figure 3: As applications such as spam or virus filters on mail servers need constant access to files, it typically makes sense to invest in a RAM disk for the filter software working directory on the server.

7. Get rid of any logfiles you don't need! Logging costs time. Even a single logfile that nobody needs is one too many. If you log on external disks, make sure you use an extremely fast SAN; NFS will tend to be a bottleneck.

8. Always use sendfile! Sendfile is a system call that delegates the passing of files from network sockets to the kernel. This saves memory (by doing without a read buffer), and is quicker at the same time. Apache will use sendfile if you enable *EnableSendfile*.

9. Be aware of MMAP! MMAP support, via the *mod_mmap_static* module, gives Apache the ability to access files like contiguous memory space, which in turn is good for performance.

10. Don't use internal server monitoring! Apache's self-monitoring ability (*SetHandler server-status...*) is useful for tests and debugging, but make sure you disable it after completing your tests.

Mail Servers

If your mail server is threatening to buckle under the load, you definitely need to sort our your priorities. First of all, you have to make sure that the system stays stable and works effectively despite the heavy load. Don't even think of optimizing for more speed until you have achieved stability. Some practical tips will help you manage the spikes.

1. Limit the number of instances! The default value in the Postfix *master.cf* file sets the maximum instance count to 100. Depending on the version and built-in capabilities, an instance can consume about 3 MB RAM, quickly leading to an out-of-memory condition on a server with restricted resources, and thus to a system crash. Spam and virus filters also increase memory requirements if you have multiple parallel instances running.

If swapping is slowing your server down, it makes sense to reduce the number of instances. Many parallel instances on an overloaded system will just interfere with one another, severely affecting the data throughput.

2. Help the spam filter with a RAM disk! A spam and virus filter on mail servers, such as Amavisd New or Spamassassin, often cause bottlenecks. They generate a heavy CPU and I/O load, and thus impact the total throughput. In this case, swapping */var/spool/amavis/tmp* out to a RAM disk might help. The improved performance means that the server can now handle 14, instead of the seven instances recommended by the vendor [4].

Avoid Roundabout Routes

3. Cache DNS requests! Mail servers rely on DNS, and they have to handle countless requests. Which are the MX servers in the domain? Does the sender's domain really exist? Is the client on your RBL list? A caching DNS server entry in */etc/resolv.conf* will save valuable milliseconds in high volume scenarios.

4. Avoid roundabout routes! The typical approach is to forward mails from Postfix to the spam and virus filters, which in turn hand them back to Postfix. Add more rounds if you use other appliances.

It is preferable to avoid handing messages from virus or spam filters back to Postfix and to pass them on to the next appliance instead. If the mail chain handles only incoming mail, the last appliance in the chain can forward the emails directly to an internal mail server instead of handing them back to Postfix.

Check Responsibilities

5. Only respond if it is your responsibility! If you accept mails that you can't deliver, you are forced to bounce and return the messages; this is a clear waste of resources. Use *local_recipient_maps* and *relay_recipient_maps* to tell Postfix to accept mails for existing accounts only. This avoids unnecessary load when spammers are just trying out addresses.

The same thing applies to source addresses: if the domain specified in the header does not exist, the message must be spam. And there is no way you can respond. To improve total performance, *reject_unknown_sender_domain* performs a DNS lookup to validate the source domain before the server accepts the message.

6. Use only local files in Postfix lookup tables! No matter how convenient MySQL or LDAP-based user or domain management may be, the effect it has on Postfix performance is negative. A lookup table in *hash* or preferably *btree* format is far quicker. It makes sense to code a script that writes the updated user data from the MySQL or LDAP table to a local file on the server every thirty minutes.

7. Keep pesky clients at bay with rate limiting! If you discover that an individual client is overtaxing the mail server, or if an attack is in progress, rate limiting via the *smtpd_client_connection_rate_limit* parameter will prevent this from

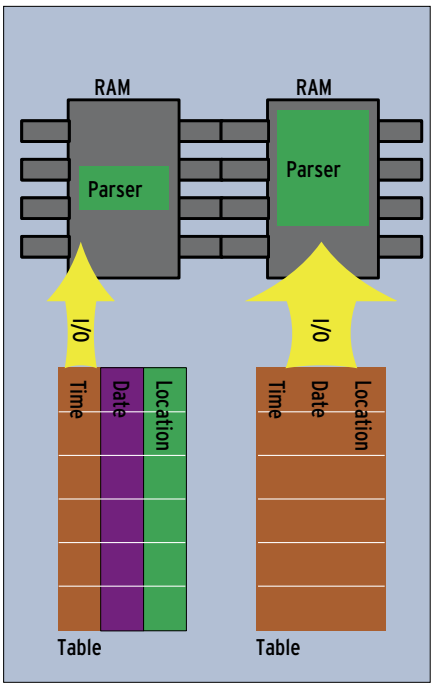

Figure 4: Partitioned tables (supported by MySQL 5.1) avoid the need for the parser to read the whole table to process requests for data from just one or two columns.

affecting your mail traffic. A firewall can restrict the maximum connect count, and thus prevent hackers on compromised clients tunneling back hundreds of connections.

8. Don't waste time with problematic mails! If you have an outgoing mail traffic jam, this could be due to Postfix wasting resources on a large volume of undeliverable mail. *maximum_backoff_time* sets the time that Postfix will wait before attempting to redeliver. Increasing this value gives you more cycles to complete a first try, instead of launching into a series of repeats that are likely to fail. As an alternative, you could set the *fallback_relay* parameter, which swaps problematic messages out to another machine that does the dirty work for the mail server.

Database Servers

The quality of the SQL queries, the design of the database, and the server configuration can considerably influence database performance. The following tips will help you boost your database server's performance.

1. Choose the right indexes! The indexes are one of the most important things about a database; the server's response times depend to a great extent on their quality. A B*TREE index (the default index type for many databases) should be used if the indexed column can hold many different values. The search tree for this index type will grow more slowly than any other. For columns with just a few different values (such as product groups), a bitmap type index is preferable for Oracle, or a similar type for other databases.

For tables with just a few rows, *TABLE ACCESS (FULL)* (or *FULL TABLE SCAN* in MySQL) will be faster than index-based access. If many queries use functions such as *UPPER(column xyz)*, an index of the function results will give you improved performance, assuming the database engine you are using supports function-based indexes [5].

2. Always delete unneeded indexes! The Oracle Optimizer does not use indexes that are not required by a statement. No matter whether an SQL statement uses them or not, the SQL engine will still load every single index that you define for a table. This costs I/O resources and CPU load.

3. Avoid fragmented indexes! B*TREE indexes are prone to fragmentation over time, due to table updates or inserts, and this really slows your queries down. You can issue an *ANALYZE INDEX index name VALIDATE STRUCTURE* in Oracle to discover the fragmentation status. *ALTER INDEX index name REBUILD ONLINE* will rebuild the index on the fly. In MySQL, *ANALYZE TABLE* and *OPTIMIZE TABLE* will rebuild a fragmented index.

The Right Expression

4. Optimize your SQL statements! SQL is a flexible language, and many roads lead to Rome. However, approaches very often differ with respect to I/O resource usage. *EXPLAIN PLAN* in Oracle or *EXPLAIN* in MySQL can help you optimize: these commands explain how the SQL engine performs a query, how it uses indexes, and how many interim results the database will generate. MySQL 5.1 or newer gives you an *EXPLAIN PARTITIONS* feature to help you analyze the runtime performance of partitioned tables [6].

5. Make good use of your database's soft parsing feature! The database engine parses SQL queries literally. The engine will not relate queries that differ with respect to a single literal (for example *SELECT...WHERE x = 100...* and *SELECT...WHERE x = 200...*). Thus a copy of each of these queries is deposited in the shared memory pool.

If queries which differ by a single literal reoccur, the database will store a hundred copies in the memory pool (hard parsing). This can lead to memory fragmentation; more RAM will probably not help. However, if your query uses a variable in the *WHERE* clause instead of a literal (command *BIND SQL*), the engine will store the SQL query along with the variable in RAM and not change it when it is reused (soft parsing).

Divide and Conquer

6. Restrict the size of your tables. Tables with millions of records consume vast amounts of I/O resources. One way of reducing this consumption is to divide large tables up into several blocks. Oracle 8 and MySQL 5.1 or newer support partitioning for tables. If you have SQL statements that only reference certain columns in their *WHERE* clauses, the parser only has to query the corresponding partitions (Figure 4). Partitioning can be a big help with batch jobs that update part of a table.

7. Keep OLTP and DSS separate! Different types of database use mean different resource requirements: Decision Support Systems (DSS, [7]) such as research or search systems typically perform many select and sort operations, but very few insert, update, or delete actions.

In contrast to this, an Online Transaction Processing System (OLTP, [8]) such as an online ordering or content management system, will typically serve hundreds of users at the same time, and all of them can modify the data. Thus, the index tables will change constantly.

Choosing the right index is vital to performance. If a database server is deployed in both scenarios, you will need to prioritize CPU usage by application type. The easiest approach to doing so with Oracle is to use the Resource Manager [9].

8. Use persistent connections and the shared server! Reestablishing non-persistent connections consumes valuable resources. On the other hand, Oracle spawns a process for every connection on a dedicated server and keeps the process in memory, even if the user isn't actually accessing the database. For this

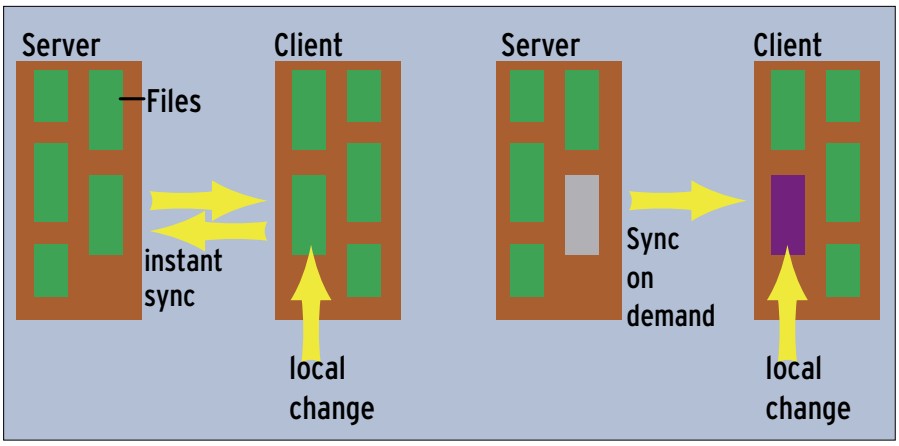

Figure 5: Opportunistic locks remove the need to sync clients and servers for write access with just one user accessing the file. If another user needs access, the server will remove the lock, and the client will return the modified file.

reason, Oracle provides a shared server solution that keeps the numbers of processes to a minimum by managing connections and queries in a large pool. This helps save resources, especially on OLTP systems.

9. Pay attention to timing with batch jobs! Batch jobs should only run at times when access levels are low, such as at night.

10. Check the number of connections! Separate application and web servers from database servers for the most effective approach. Whatever you do, make sure the maximum number of processes on both sides are in agreement; if the maximum number of processes on the web server is higher than for the database, the database might be swamped with connections that it can't handle due to the large numbers of processes involved.

The Apache *MaxClients*, *KeepAlive*, *MaxSpare-Servers*, and *Max-Requests-PerChild* directives must match your Oracle settings. The *SESSION* and *PROCESS* parameters, and the decision whether or not to use a shared server, will influence the maximum number of supported connections. MySQL uses *max_connections* and *max_user_connections* to set these limits.

Samba

If your Samba server does not serve up your data as fast as you would like, opening and saving files can be extremely time-consuming, and this will affect productivity throughout the enterprise. These tips will help you fend off bottlenecks.

1. Never touch your Samba configuration! Samba comes with a perfect default configuration that gives Windows clients the best possible performance. If you experience performance problems despite this, you should start by removing any options you do not need from the *smb.conf* configuration file before going on to change options in phase two.

2. Be generous with RAM! 2 to 3 MB of physical RAM per logged on user are the minimum for a powerful Samba server. The system can use every single byte on top of this as cache memory for ever better performance.

3. Check out your network! If file transfers to and from the Samba server are slow, use FTP to test. FTP is a very simple protocol that uses TCP-only data streams and thus gives you the maximum transfer rate over your network.

Useful Helpers

4. Enable oplocks! An opportunistic lock, or oplock for short, is not designed to reserve files for exclusive use by one user. Instead, an oplock gives the client the ability to cache the file content. The server assures the client that nobody else will be able to access the file. If a second user opens the file, the server will contact the client, and the client will release the oplock (Figure 5). The client does not send its changes to the server until this point, and this means considerable network bandwidth savings.

5. Launch Winbind! Without Winbind, the Samba daemon has to set up a connection for every new user, and this causes an overhead of 40 to 60 IP packets. Winbind reduces this to three packets by keeping the connection to the domain controller open.

6. Don't distinguish between upper and lower case. As Windows filenames are not case-sensitive, but Unix filenames are, Samba has to make sure that a file called *Test.txt* does not exist if Windows wants to create a file called *test.txt*. For directories with over a thousand entries, the resulting scan is bound to be heavy on resources. If possible, restrict any directories managed by Samba to a small number of files. If this is not an option, disable the scan by disabling *case sensitive = yes*, *preserve case = no*, and *default case = lower*. Samba will then display all filenames as lower case.

A Clear Conscience

It's 6.30 pm. the response times on the web server with the link from Slashdot are acceptable. The mass mailing campaign went off smoothly, and the database has evaluated the survey well before the deadline. After applying the tips in this article to optimize the server, it's time for the overworked system administrator to head for home, secure in knowing the systems can handle the load. ■

INFO

[1] Postfix Anvil Server: *http://www.postfix.org/anvil.8.html*

[2] Cband: *http://cband.linux.pl*

[3] Apache mod_mem_cache module: *http://httpd.apache.org/docs/2.0/mod/mod_cache.html*

[4] Performance metrics for virus and spam checks: *http://www.heinlein-support.de/upload/martinec.pdf*

[5] Function-based indexes with Oracle: *http://download-uk.oracle.com/docs/cd/B14117_01/server.101/b10752/data_acc.htm#2185*

[6] EXPLAIN PARTITIONS in MySQL 5.1: *http://dev.mysql.com/doc/refman/5.1/en/explain.html*

[7] Decision Support Systems (Wikipedia): *http://en.wikipedia.org/wiki/Decision_Support_System*

[8] Online Transaction Processing (Wikipedia): *http://en.wikipedia.org/wiki/Online_Transaction_Processing*

[9] Oracle Resource Manager: *http://download-uk.oracle.com/docs/cd/B14117_01/server.101/b10739/dbrm.htm#i1010776*