PostgreSQL Magazine
News and stories from the Postgres Community

Performance Tuning :
To infinitity..... and beyond !

Special Edition

Interview : Bruce Momjian
Case Study : French Social Security System
Waiting for 9.1 : Extensions
Opinion : Oracle and Open Source

May 2011
#0

PGMag Team

Editor :

Writers :

Reviewers :

Editorial

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut a sapien. Aliquam aliquet purus molestie dolor. Integer quis eros ut erat posuere dictum. Curabitur dignissim. Integer orci. Fusce vulputate lacus at ipsum. Quisque in libero nec mi laoreet volutpat. Aliquam eros pede, scelerisque quis, tristique cursus, placerat convallis, velit. Nam condimentum. Nulla ut mauris. Curabitur adipiscing, mauris non dictum aliquam, arcu risus dapibus diam, nec sollicitudin quam erat quis ligula. Aenean massa nulla, volutpat eu, accumsan et, fringilla eget, odio. Nulla placerat porta justo. Nulla vitae turpis. Praesent lacus.

Vivamus neque velit, ornare vitae, tempor vel, ultrices et, wisi. Cras pede. Phasellus nunc turpis, cursus non, rhoncus vitae, sollicitudin vel, velit. Vivamus suscipit lorem sed felis. Vestibulum vestibulum ultrices turpis. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Praesent ornare nulla nec justo. Sed nec risus ac risus fermentum vestibulum. Etiam viverra viverra sem. Etiam molestie mi quis metus hendrerit tristique.

Suspendisse nibh. Nunc vulputate leo id urna. Donec dictum. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Suspendisse dictum, magna consectetuer hendrerit volutpat, sapien felis faucibus justo, ac dictum lacus pede in metus. Nam commodo. Sed consequat, leo pretium sagittis congue, ante nunc laoreet nisl, ac aliquam risus tellus commodo elit. Cras at elit. Pellentesque suscipit erat vitae mauris. Sed iaculis eros vitae mauris.

Upcoming Events

PgEast Conference 2011
22-25 march 2011
New York CIty, United States of America

This conference is the followup PgWest 2010. The venue, the Hotel Pennsylvania and an increase to four days instead of three promise to make this the largest PostgreSQL Conference of 2011.

PgEast Conference 2011
22-25 march 2011
New York CIty, United States of America

This conference is the followup PgWest 2010. The venue, the Hotel Pennsylvania and an increase to four days instead of three promise to make this the largest PostgreSQL Conference of 2011.

The PostgreSQL Community with international-class Conference almost every month !
all around the world

PgEast Conference 2011
22-25 march 2011
New York CIty, United States of America

This conference is the followup PgWest 2010. The venue, the Hotel Pennsylvania and an increase to four days instead of three promise to make this the largest PostgreSQL Conference of 2011.

PgEast Conference 2011
22-25 march 2011
New York CIty, United States of America

This conference is the followup PgWest 2010. The venue, the Hotel Pennsylvania and an increase to four days instead of three promise to make this the largest PostgreSQL Conference of 2011.

PgEast Conference 2011
22-25 march 2011
New York CIty, United States of America

This conference is the followup PgWest 2010. The venue, the Hotel Pennsylvania and an increase to four days instead of three promise to make this the largest PostgreSQL Conference of 2011.

PgEast Conference 2011
22-25 march 2011
New York CIty, United States of America

This conference is the followup PgWest 2010. The venue, the Hotel Pennsylvania and an increase to four days instead of three promise to make this the largest PostgreSQL Conference of 2011.

Announces

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut a sapien. Aliquam aliquet purus molestie dolor. Integer quis eros ut erat posuere dictum. Curabitur dignissim. Integer orci. Fusce vulputate lacus at ipsum. Quisque in libero nec mi laoreet volutpat. Aliquam eros pede, scelerisque quis, tristique cursus, placerat convallis, velit. Nam condimentum. Nulla ut mauris. Curabitur adipiscing, mauris non dictum aliquam, arcu risus dapibus diam, nec sollicitudin quam erat quis ligula. Aenean massa nulla, volutpat eu, accumsan et, fringilla eget, odio. Nulla placerat porta justo. Nulla vitae turpis. Praesent lacus. Nam iaculis blandit purus. Mauris odio nibh, hendrerit id, cursus vel, sagittis a,

Product News

Bruce Momjian : 15 years of PostgreSQL Hacking

PostgreSQL is the most advanced and flexible Open Source SQL database today. With this power and flexibility comes a problem. How do the PostgreSQL developers tune the default configuration for everyone? Unfortunately the answer is they can't.

CL: First of all, how do you pronounce PostgreSQL?

BM: It's called Post-gres-cue-el (not Postgre-es-cue-el). It's a very unusual name. It's very hard to pronounce. Somebody described it as anti-marketing in a sense that it's so confusing that people maybe don't even want to say it :-)

The original database software was called Postgres. When SQL was added to it and its development was taken onto the Internet, we added SQL at the end of the name. It looks like Postgreee plus something at the end, it does confuse people. I really do apologize for that :-)

CL: You are the vice president of Great Bridge, now would you call yourself a developer?

BM: My job with Great Bridge involves several things.

First, I'm to continue steering Postgres development as I have in the past four years. That means encouraging developers, helping to keep the group together, sort of holding the group. And obviously I'm still involved with development to some extent, adding features, fixing things. So in one sense my job is to just continue doing what I've been doing for the past four years. The second part of my job is to help steer Great Bridge as I have Postgres. That means helping them understand Postgres community, helping them work with the open source community.

So although my title is a one that's very simple, it has sort of two parts. One part is the open source part, the second part is trying to help the company to develop in a way that we think is best.

CL: How many hours do you actually write code a day?

BM: When I'm talking about the part of my job that has to do with the open source project, I spend about an hour and a half a day reading email, responding to questions, making suggestions, maintaining the code, applying patches that people have presented on the mailing list. Additionally, I do coding when something needs fixing. When there is something we need to patch or something somebody submitted that doesn't quite work anymore, I'd be coding and clean it up trying to fix it.

But I have not had a lot of time to do development since I started writing the book ("PostgreSQL: Introduction and Concepts"). I started writing my book in October of 1999, and frankly, since then up to today it's consumed almost all of my time. In fact, just as the book finished in early October (of 2000), I was employed by Great Bridge, and Great Bridge is just starting now, so I have to give them the attention to get them started and make sure they are on the right track.

Certainly, in the future, I'm going to have time to do more development.

CL: Do you have enough time to play with your children?

BM: I've worked at home for the past eight or nine years. So I was home all the time in my previous job as a database software developer.

With Great Bridge, I'm doing more traveling. Obviously I'm here in Japan. But when I'm not traveling, I'm home. So my family is happy that this new job is not taking me away from being there.

So, yeah, I'm home a lot and they come down and visit and... I see a lot of them :-)

CL: Would you say that PostgreSQL is now ready for e-business?

BM: We are already ready, but we know that we can do even more to make it better.

Certainly, it's been used in e-business already by a large number of people. We have people writing shopping carts software that uses our transactional model. The features we have make it very natural, useful, for all steady places where databases would be used.

Now, there are some features that we don't have. We believe that we need the ability to recover from a hard disk crash with the log file that we are working on for 7.1. We know that replication is something that a lot of people are looking for.

So we are continuing to target the areas that we think makes us even more attractive to e-business sites. We are focusing on that.

CL: There were benchmarks this year that said PostgreSQL was faster than commercial ones or, even MySQL. Although I think Postgres has had a good reputation of having advanced features such as the powerful data model, rich data types and high extensibility, I don't think it has had a reputation of being very fast. What happened to PostgreSQL in regard to the speed improvement?

BM: We have been working on improving performance since we started, well, since we took the code from Berkeley. And I'd say within six months, once we've had the code stable, where it was working and it wasn't

are people who have been used to analyzing performance in different ways. All look at the code, and all add the things that they felt would improve performance.

For example, several years ago, I, over period of probably two weeks, went through the code and inlined some of the function calls that was called ten thousand times, or something like that. I made a macro out of it, and I got six percent speed improvement. And then on start-up, we were doing a fork and exec which we didn't need to do. Fork was enough and we got rid of exec. All of the sudden, we got a small speed improvement there.

Bruce We did little things along the way. Tom Lane has gone through and analyzed performance in some other ways he is used to doing, and he's found many areas where he could improve performance. We've had other people who looked at memory manager, "We could improve performance a little bit if we did this," you know.

So what we've had is not one grand person coming in. Performance tuning, or performance analysis, is really a complex thing. Certain people look for certain things in performance, and other people look for different things. We are really glad having one guy who says "I'm wondering, I'm going to look at code this way," somebody else who comes along and would say "I'm going to improve performance this way," and Tom Lane who would look at another way. What we've found is that... all these people looking at it at different times along the way, adding three percent here, six percent here, two percent here, five percent here, all of the sudden, "Wow, we're really fast!"

There wasn't really one thing. Even in the benchmarks that you are talking about, where we've had improvement from 6.5 to 7.0, there really wasn't much emphasis. We were all very surprised because we didn't do much in terms of performance between 6.5 and 7.0. I changed the way that cache looked up system table information, and Tom Lane added some changes to memory

management of some other changes... I think we had 13% change between releases, but we weren't really concentrated on performance. We added a few little things that we thought might improve performance, but we didn't anticipate that.

To extend that, when I first saw the charts of performance us against all the other databases, they showed it to me in California... I was talking with somebody, and I was looking at the chart. I was looking at the one on the bottom. There was something about it, so I was saying "I wonder why this looks this way," and so forth. Someone looked at it and said "Well, this looks pretty good. The two are separate, but they are not that far apart." We were leaving the room and we were upstairs. I was talking with somebody about this chart we've been looking at, and saying "Oracle's graph looks like this and Postgres's goes up or down..." and the person I was talking to, Tom Lane I think, said "Look! The line at the top, the better database is Postgres!!"

It wasn't Oracle, and I was kind of shocked, because when I looked at the chart, legends that told me which line was which were so small, that I just assumed that Oracle was the better one, we were kind of close to them, but below them. And it turned out as a shock, that we were the one that was better, Oracle was the one that was worse.

That's an indication how surprised we've been by these performance measurements. We have received anecdotal information from people saying that Postgres was faster than Excel, but I believe it was not until we saw a systematic test of a really nationally recognized benchmark like AS3AP, that we really started to say "Wow, there really must be something...!"

CL: Where would you position yourself among these three: free software movement (like Richard Stallman), open source movement (like Eric Raymond), interested in free beer (like Linus Torvalds :-)?

Bruce BM: I can't compare myself directly to individuals, but I can tell you specifically how we view our software in terms of distribution and legal issues.

We don't have any trouble with people using our software to make money. A lot of us, myself included, spent years developing commercial proprietary software. The reason it was proprietary was because it was made for the limited market. You did custom software. Somebody called you and said "Can you add this program for me?" and there was no sense in open sourcing, because nobody else wanted it. It's just for that customer. It's like making shoes individually. When you make a shoe, you cut the size of a foot, and make the shoe

Performance Tuning PostgreSQL

PostgreSQL is the most advanced and flexible Open Source SQL database today. With this power and flexibility comes a problem. How do the PostgreSQL developers tune the default configuration for everyone? Unfortunately the answer is they can't.

The problem is that every database is not only different in its design, but also its requirements. Some systems are used to log mountains of data that is almost never queried. Others have essentially static data that is queried constantly, sometimes feverishly. Most systems however have some, usually unequal, level of reads and writes to the database. Add this little complexity on top of your totally unique table structure, data, and hardware configuration and hopefully you begin to see why tuning can be difficult.

The default configuration PostgreSQL ships with is a very solid configuration aimed at everyone's best guess as to how an "average" database on "average" hardware should be setup. This article aims to help PostgreSQL users of all levels better understand PostgreSQL performance tuning.
Understanding the process

The first step to learning how to tune your PostgreSQL database is to understand the life cycle of a query. Here are the steps of a query:

1. Transmission of query string to database backend
2. Parsing of query string
3. Planning of query to optimize retrieval of data
4. Retrieval of data from hardware
5. Transmission of results to client

The first step is the sending of the query string (the actual SQL command you type in or your application uses) to the database backend. There isn't much you can tune about this step, however if you have a very large queries that cannot be prepared in advance it may help to put them into the database as a stored procedure and cut the data transfer down to a minimum.

Once the SQL query is inside the database server it is parsed into tokens. This step can also be minimized by using stored procedures.

The planning of the query is where PostgreSQL really starts to do some work. This stage checks to see if the query is already prepared if your version of PostgreSQL and client library support this feature. It also analyzes your SQL to determine what the most efficient way of retrieving your data is. Should we use an index and if so which one? Maybe a hash join on those two tables is appropriate? These are some of the decisions the database makes at this point of the process. This step can be eliminated if the query is previously prepared.

Now that PostgreSQL has a plan of what it believes to be the best way to retrieve the data, it is time to actually get it. While there are some tuning options that help here, this step is mostly effected by your hardware configuration.

And finally the last step is to transmit the results to the client. While there aren't any real tuning options for this step, you should be aware that all of the data that you are returning is pulled from the disk and sent over the wire to your client.

Minimizing the number of rows and columns to only those that are necessary can often increase your performance.
General Tuning

There are several postmaster options that can be set that drastically affect performance, below is a list of the most commonly used and how they effect performance:

* max_connections = <num> — This option sets the maximum number of database backend to have at any one time. Use this feature to ensure that you do not launch so many backends that you begin swapping to disk and kill the performance of all the children. Depending on your application it may be better to deny the connection entirely rather than degrade the performance of all of the other children.

* shared_buffers = <num> — Editing this option is the simplest way to improve the performance of your database server. The default is pretty low for most modern hardware. General wisdom says that this should be set to roughly 25% of available RAM on the system. Like most of the options I will outline here you will simply need to try them at different levels (both up and down) and see how well it works on your particular system. Most people find that setting it larger than a third starts to degrade performance.

* effective_cache_size = <num> — This value tells PostgreSQL's optimizer how much memory PostgreSQL has available for caching data and helps in determing whether or not it use an index or not. The larger the value increases the likely hood of using an index. This should be set to the amount of memory allocated to shared_buffers plus the amount of OS cache available. Often this is more than 50% of the total system memory.

* work_mem = <num> — This option is used to control the amount of memory using in sort operations and hash tables. While you may need to increase the amount of memory if you do a ton of sorting in your application, care needs to

be taken. This isn't a system wide parameter, but a per operation one. So if a complex query has several sort operations in it it will use multiple work_mem units of memory. Not to mention that multiple backends could be doing this at once. This query can often lead your database server to swap if the value is too large. This option was previously called sort_mem in older versions of PostgreSQL.

* max_fsm_pages = <num> — This option helps to control the free space map. When something is deleted from a table it isn't removed from the disk immediately, it is simply marked as "free" in the free space map. The space can then be reused for any new INSERTs that you do on the table. If your setup has a high rate of DELETEs and INSERTs it may be necessary increase this value to avoid table bloat.

* fsync = <boolean> — This option determines if all your WAL pages are fsync()'ed to disk before a transactions is committed. Having this on is safer, but can reduce write performance. If fsync is not enabled there is the chance of unrecoverable data corruption. Turn this off at your own risk.

* commit_delay = <num> and commit_siblings = <num> — These options are used in concert to help improve performance by writing out multiple transactions that are committing at once. If there are commit_siblings number of backends active at the instant your transaction is committing then the server waiting commit_delay microseconds to try and commit multiple transactions at once.

* random_page_cost = <num> — random_page_cost controls the way PostgreSQL views non-sequential disk reads. A higher value makes it more likely that a sequential scan will be used over an index scan indicating that your server has very fast disks.

If this is still confusing to you, Revolution Systems does offer a PostgreSQL Tuning Service

Note that many of these options consume shared memory and it will probably be necessary to increase the amount of shared memory allowed on your system to get the most out of these options.

Hardware Issues

Obviously the type and quality of the hardware you use for your database server drastically impacts the performance of your database. Here are a few tips to use when purchasing hardware for your database server (in order of importance):

* RAM — The more RAM you have the more disk cache you will have. This greatly impacts performance considering memory I/O is thousands of times faster than disk I/O.

* Disk types — Obviously fast Ultra-320 SCSI disks are your best option, however high end SATA drives are also very good. With SATA each disk is substantially cheaper and with that you can afford more spindles than with SCSI on the same budget.

* Disk configuration — The optimum configuration is RAID 1+0 with as many disks as possible and with your transaction log (pg_xlog) on a separate disk (or stripe) all by itself. RAID 5 is not a very good option for databases unless you have more than 6 disks in your volume. With newer versions of PostgreSQL you can also use the tablespaces option to put different tables, databases, and indexes on different disks to help optimize performance. Such as putting your often used tables on a fast SCSI disk and the less used ones slower IDE or SATA drives.

* CPUs — The more CPUs the better, however if your database does not use many complex functions your money is

best spent on more RAM or a better disk subsystem.

In general the more RAM and disk spindles you have in your system the better it will perform. This is because with the extra RAM you will access your disks less. And the extra spindles help spread the reads and writes over multiple disks to increase throughput and to reduce drive head congestion.

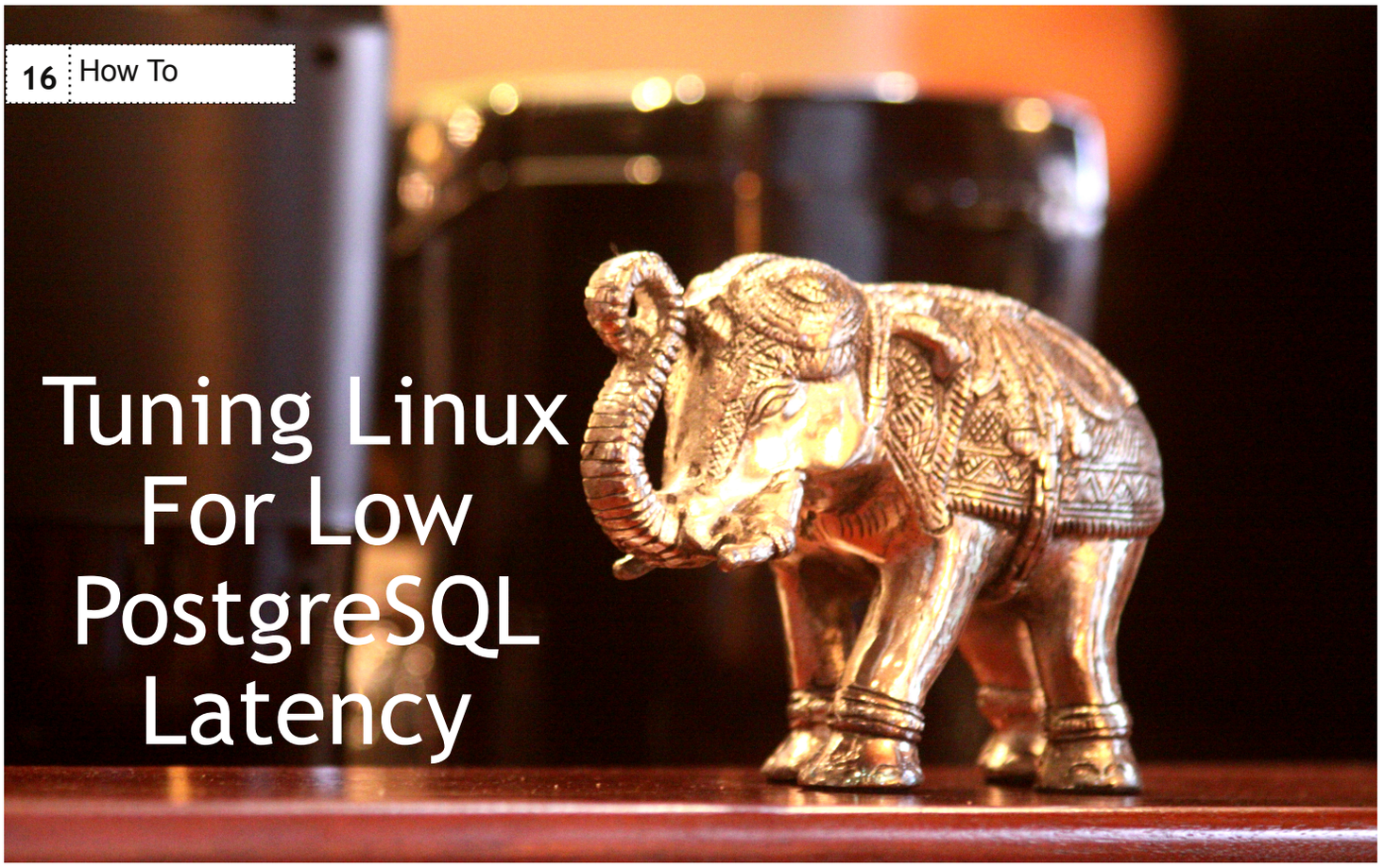
Another good idea is to separate your application code and your database server onto different hardware. Not only does this provide more hardware dedicated to the database server, but the operating system's disk cache will contain more PostgreSQL data and not other various application or system data this way.

For example, if you have one web server and one database server you can use a cross-over cable on a separate ethernet interface to handle just the web server to database network traffic to ensure you reduce any possible bottlenecks there. You can also obviously create an entirely different physical network for database traffic if you have multiple servers that access the same database server.

Application Development

There are many different ways to build applications which use a SQL database, but there are two very common themes that I will call stateless and stateful. In the area of performance there are different issues that impact each.

Stateless is typically the access type used by web based applications. Your software connects to the database, issues a couple of queries, returns to results to the user, and disconnects. The next action the



Tuning Linux For Low PostgreSQL Latency

One of the ugly parts of Linux with PostgreSQL is that the OS will happily cache up to around 5% of memory before getting aggressive about writing it out. I've just updated a long list of pgbench runs showing how badly that can turn out, even on a server with a modest 16GB of RAM. Note that I am intentionally trying to introduce the bad situation here, so this is not typical performance. The workload that pgbench generates is not representative of any real-world workload, it's as write-intensive as it's possible to be.

Check out test set 5, which is running a stock development version PostgreSQL 9.1. Some the pauses where the database is unresponsive during checkpoints, as shown by the max_latency figure there (which is in milliseconds), regularly exceed 40 seconds. And at high client counts you can see >80 seconds of the database completely stalled. Check out a sample of a really bad one. See the dead spot between 26:00 and 27:00 where no transactions are processed? That's a bad thing.

It used to be possible to improve these by tuning Linux's dirty_ratio and dirty_background ratio parameters. I wrote an article called The Linux Page Cache and pdflush covering that topic a few years back. Nowadays system RAM is so large that even the minimum settings possible there are caching way too much. A better UI was introduced in kernel 2.6.29. Now you can set these in bytes instead, which allows much smaller settings, using dirty_bytes and dirty_background_bytes.

You can see what happens when you tune those down by looking at test set #7 [Note that this also includes a work in progress PostgreSQL patch to fix some other bad behavior in this area, introduced in set #6] I set the new dirty_* parameters to 64MB/128MB, trying to be below the 256MB battery-backed cache in the RAID card. The parts that improved are quite obvious. Now maximum latency on the smaller tests drops to <10 seconds. And the worst one shown is just over 20 seconds now. Both of these are about 1/4

of the worst-case latency shown before I tweaked these parameters.

There is a significant drop in transactions/second, too, but not necessarily an unacceptable one. Across the whole set the averages were scale=500, tps=690 and scale=1000, tps=349 before; now it's scale=500, tps=611 and scale=1000, tps=301. Batching up work into larger chunks always gives higher transaction rates but lower latency. That part would be a completely reasonable trade-off in a lot of situations: 1/4 the worst-case latency for a 10-15% drop in TPS.

Unfortunately I discovered a bigger downside here too. In between each pgbench test, there is some basic table clean up done to try and make the tests more repeatable. Part of that is doing a VACUUM of the database. I noticed that this test series took way longer to finish than any previous one. The figure you can compare reasonably here is to ask "what's the minimum time seen to clean the database up?". That varies a bit from test to test, but the minimums across a larger set are quite consistent. It takes a certain amount of time to do that job, and the fastest such cleanup job is a pretty stable figure here at a given database size.

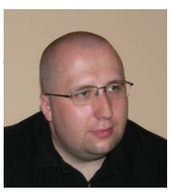
Here are those figures from the last 3 test sets:

```
 set | scale |  min_setup_time
-----+-------+-----------------
   5 |   500 | 00:03:41.220735
   5 |  1000 | 00:06:47.818328
   6 |   500 | 00:03:33.139611
   6 |  1000 | 00:06:41.154474
   7 |   500 | 00:06:06.56808
   7 |  1000 | 00:10:14.010876
```

You can see that test sets 5 and 6 both took around 3.5 minutes to cleanup a scale=500 database, and 6.75 minutes for scale=1000. Dialing down the cached memory Linux was keeping around increased those times to 6 minutes and 10 minutes instead. That's 71% and 48% longer, respectively, than those operations took with a large amount of write caching managed by the kernel.

Now that is a much harder price to pay for decreased latency. Looks like some of the server-side work planned to try and reduce these checkpoint spikes is still completely relevant even on a kernel that has these knobs available.

About the author



Hubert "Depesz" Lubaczewski is a Database Architect at OmniTI. His blog (http://www.depesz.com/) is dedicated to the new features of the forthcoming version of PostgreSQL.

The original article is available at http://is.gd/qdC51B

PostgreSQL at CNAF : 1 billion SQL queries per day

Bull modernizes CNAF's core business applications

* Global consolidation project designed to guarantee performance and openness
* The proven robustness of state-of-the-art Open Source technology, capable of managing a critical application workload
* Migration to a PostgreSQL Open Source database, operating on a uniquely large scale

The Caisse Nationale d'Allocations Familiales (CNAF) – the family branch of the French social security system, which provided some €69 billion of benefits to 11 million claimants in 2009 – has turned to Bull to migrate its Interel-RFM2 database running under gcos8 to PostgreSQL. The decision is a reaffirmation of CNAF's commitment to move towards Open Source solutions. As a result of this, every month PostgreSQL is involved in the payment of €3 billion in benefits to CNAF claimants. The project, which is now fully up and running, is an integral part of the program to transform the information systems of this highly sensitive organization, which will soon have to bring on board new initiatives such as the RSA (Revenu de Solidarité Active), the French government's minimum income guarantee for low-income households.

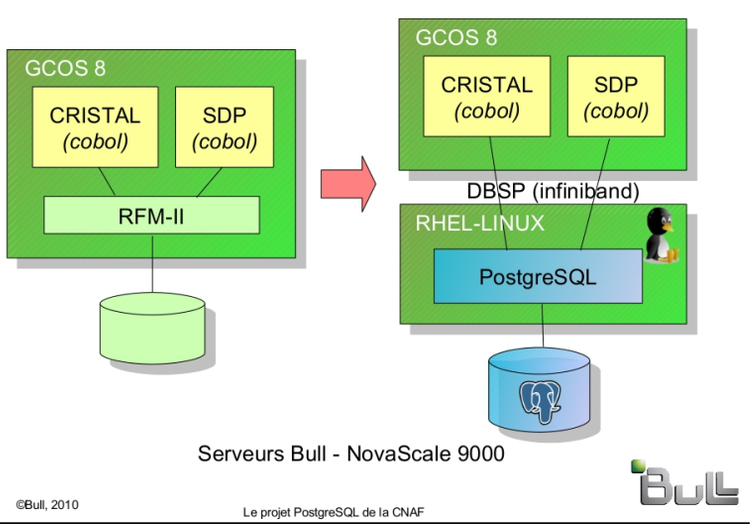
Meeting the needs for performance and scalability

To improve the quality of its services, against a backdrop of cost constraints, CNAF has put the emphasis on the search for better performance and more powerful systems. To achieve this, it decided to modernize two of its strategic and rapidly evolving applications, CRISTAL and SDP, which together account for 20 million lines of application software code and are central to its core business of calculating people's entitlements and paying benefits. The choice of a new database that meets CNAF's criteria in terms of performance and openness was an obvious step, with PostgreSQL proving to be the best option.

As Marc Pavie, Deputy Director of Information Systems at CNAF, explained: "The choice of PostgreSQL reiterates our commitment to Open Source. With Bull as prime contractor, we migrated without any difficulty. With PostgreSQL we are now benefitting from new functionality which is contributing to the optimization of our information system. But most importantly the solution offers levels of robustness and performance largely in line with our production imperatives and challenges at a national level relating to the quality of our services."

Over a billion SQL queries run every day

The migration project started in October 2008,



following a feasibility study carried out by Bu and a prototype built in collaboration with CNAF's teams. An extensive testing phase established a sound basis for the implementation, which took place over the nine months to April 2010.

Bull contributed to this project by providing:

* Project management support
* All the technical expertise needed to design and scope the system architectures
* Training for the various teams on the new RDBMS
* Development of the data migration tool
* Assistance with adapting the code and testing
* The novascale platform, including PostgreSQL
* Overall support for the solution, with the option to escalate any problems to level 3 via the Bull center of expertise in Phoenix, USA.

In total, it took just 18 months to migrate the 168 databases involved, representing a total of 4 Terabytes of data. Since then, almost a billion SQL queries are being run every day. Regular administrative tasks have been automated and system supervision is carried out using an open software solution, Nagios. In addition, the architecture that has been implemented features dual high-availability, with a remote business recovery site and locally based service continuity facilities.

Several months after the implementation was completed, the PostgreSQL database is clearly proving its robustness and levels of performance. As a result, the reduction in batch processing times and transaction response times has led to improvements in the quality of the service provided.

Jean-Pierre Barbéris, CEO of Bull France, commented: "We are very proud to have supported CNAF in this project, which is unprecedented in Europe. The resulting benefits bear witness not only to the extraordinary power and robustness of Open Source tools, but also to their ability to support organizations' most helping them to achieve greater room for maneuver financially."
A novascale gcos 9010 server to run the production infrastructure

In parallel, during the first six months of 2010 CNAF has also been consolidating its various production resources running under gcos and Linux into a single Data Center equipped with a novascale gcos 9010 server, the new generation of mainframe-class Bull servers offering the best of mainframe technology combined with the benefits of the open world, especially PostgreSQL.

About the author

Hubert "Depesz" Lubaczewski is a Database Architect at OmniTI. His blog (http://www.depesz.com/) is dedicated to the new features of the forthcoming version of PostgreSQL.

The original article is available at http://is.gd/qdC51B

Extensions coming in PostgreSQL 9.1

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut a sapien. Aliquam aliquet purus molestie dolor. Integer quis eros ut erat posuere dictum. Curabitur dignissim. Integer orci. Fusce vulputate lacus at ipsum. Quisque in libero nec mi laoreet volutpat. Aliquam eros pede, scelerisque

On 8th of February, Tom Lane committed this patch:

```
Core support for "extensions", which
are packages of SQL objects.

This patch adds the server
infrastructure to support extensions.
There is still one significant loose
end, namely how to make it play nice
with pg_upgrade, so I am not yet
committing the changes that would make
all the contrib modules depend on this
feature.

In passing, fix a disturbingly large
amount of breakage in
AlterObjectNamespace() and callers.

Dimitri Fontaine, reviewed by Anssi
Kääriäinen, Itagaki Takahiro, Tom Lane,
and numerous others
```

Extensions are a way to group various database objects together to allow easy manipulation. For starters – all contrib modules became extensions. This means that installing and changing them is now much simpler. For example, adding ltree contrib module to database is now as simple as:

```
$ select '1.1'::ltree;
ERROR: type "ltree" does not exist
LINE 1: select '1.1'::ltree;
                       ^
$ create EXTENSION ltree;
CREATE EXTENSION
$ select '1.1'::ltree;
ltree
-------
1.1
```

moving it is also trivial:

```
$ drop EXTENSION ltree;
DROP EXTENSION
```

But the great stuff is that you can easily load the extension to different schema, and after loading you can easily alter it. Let's see how it works:

```
$ create extension ltree;
CREATE EXTENSION
$ create table test (z ltree);
CREATE TABLE
```

OK. So I have table with column of ltree datatype. But now I decided, I don't want ltree-related functions "polluting" public schema, and I want to move it to designated schema just for it:

```
$ create schema ltree;
CREATE SCHEMA
$ alter extension ltree set schema ltree;
ALTER EXTENSION
```

Table got modified too:

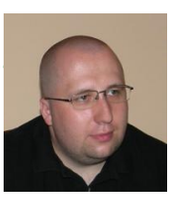
```
$ \d test
Table "public.test"
Column  | Type         | Modifiers
--------+--------------+----------
-
z       | ltree.ltree |
```

While this might not seem like a big deal, in fact it is. Having all objects grouped together means that you can upgrade them, and dump/restore is easier. As instead of dumping all functions/tables/types definitions, dump contains only:

```
=$ pg_dump | grep ltree
-- Name: ltree; Type: SCHEMA; Schema: -;
Owner: depesz
CREATE SCHEMA ltree;
ALTER SCHEMA ltree OWNER TO depesz;
-- Name: ltree; Type: EXTENSION; Schema:
-; Owner:
CREATE EXTENSION ltree WITH SCHEMA
ltree;
-- Name: EXTENSION ltree; Type: COMMENT;
Schema: -; Owner:
COMMENT ON EXTENSION ltree IS 'data type
for hierarchical tree-like structures';
z ltree.ltree
```

Benefit of this will be understood by anyone who ever tried to load dump made on some PostgreSQL to other PostgreSQL, when there were loaded extensions (modules), and paths did change. Or new version of contrib module didn't provide the same functions (some new, some old disappeared). Writing extensions looks trivial, so I assume that we will see migration of pgFoundry projects to extensions approach soon.

About the author



Hubert "Depesz" Lubaczewski is a Database Architect at OmniTI. His blog (http://www.depesz.com/) is dedicated to the new features of the forthcoming version of PostgreSQL.

The original article is available at http://is.gd/qdC51B

NoSQL ?

PG MAG Needs You !

Tips & Tricks