Choosing a non-relational database; why we migrated from MySQL to MongoDB

2009 July 25

by David Mytton

Until recently, our server monitoring application, Server Density, was running using MySQL for the backend. Although we primarily provide it as a hosted service, it has been written to work as a standalone application for customers that wish to install on their own servers. This means each customer had their own MySQL database.

We collect a lot of data – the monitoring agent reports back every 60 seconds and includes various statistics, of which the server snapshot has the most data (because it is collecting details on every running process). Over time, this results in millions of rows in the database, even for just 1 month of data, per server monitored.

Despite this huge amount of data, performance was not a problem. We were able to tune our queries and servers to ensure that results were returned quickly. The majority of queries are inserts with a minimal number of reads. We are also not doing any caching, yet. Once we reach that stage in our scaling plan there will be even fewer reads directly from the DB because the latest metric values will be stored in memory.

The problem we encountered was administrative. We wanted to scale using replication but found that MySQL had a hard time keeping up, especially with the initial sync. As such, backups became an issue, but we solved that. However, scaling MySQL onto multiple clustered servers as we plan to do in the future is difficult. You either do this through replication but that is only really suited to read-heavy applications; or using MySQL cluster. The cluster looks very good but I have read about some problems with it and was unsure of it's suitability for our needs.

The current fashion is using key/value stores, also known as non-relational database management systems (non-RDBMS) or schema-less databases. As such I did some research into the options available.

The different options

Our requirements were a relatively stable product with a decent access interface, persistent disk based storage (many are in-memory only), a good community and some documentation. Based on that, the projects I reviewed were:

- Cassandra
- CouchDB
- Hypertable
- MongoDB
- Tokyo Cabinet
- Project Voldemort

There are several good blog posts around that go into more detail for each project.

I did not consider "cloud" databases such as Amazon SimpleDB because I thought the latency would be too high.

Why we chose MongoDB

I tested most of these with real data and eventually chose MongoDB for a number of reasons:

- Very easy to install.
- PHP module available.
- Very easy replication, including master-master support. In testing this caught up with our live DB very quickly and stayed in sync without difficulty.
- Automated sharding being developed.
- Good documentation.

Implementation details

Switching from relational to non-relational is time consuming but it is not difficult. That said, there are differences that you won't necessarily be expecting. Some of these are specific to MongoDB but some will apply generally too:

Schema-less

This means things are much more flexible for future structure changes but it also means that every row records the field names. We had relatively long, descriptive names in MySQL such as timeAdded or valueCached. For a small number of rows, this extra storage only amounts to a few bytes per row, but when you have 10 million rows, each with maybe 100 bytes of field names, then you quickly eat up disk space unnecessarily. 100 * 10,000,000 = ~900MB just for field names!

We cut down the names to 2-3 characters. This is a little more confusing in the code but the disk storage savings are worth it. And if you use sensible names then it isn't that bad e.g. timeAdded -> tA. A reduction to about 15 bytes per row at 10,000,000 rows means ~140MB for field names – a massive saving.

The database-per-customer method doesn't work

MongoDB stores data in flat files using their own binary storage objects. This means that data storage is very compact and efficient, perfect for high data volumes. However, it allocates a set of files per database and pre-allocates those files on the filesystem for speed:

> Each datafile is preallocated to a given size. (This is done to prevent file system fragmentation, among other reasons.) The first file for a database is .0, then .1, etc. .0 will be 64MB, .1 128MB, etc., up to 2GB. Once the files reach 2GB in size, each successive file is also 2GB.
>
> Thus if the last datafile present is say, 1GB, that file might be 90% empty if it was recently reached.

This was a problem because MongoDB was frequently pre-allocating in advance when the data would almost never need to "flow" into another file, or only a tiny amount of another file. This is particularly the case with free accounts where we clear out data after a month. Such pre-allocation caused large amounts of disk space to be used up.

We therefore changed our data structure so that we had a single DB, thus making the most efficient use of the available storage. There is no performance hit for doing this because the files are split out, unlike MySQL which uses a single file per table.

Unexpected locking and blocking

In MongoDB, removing rows locks and blocks the entire database. Adding indexes also does the same. When we imported our data, this was causing problems because large data sets were causing the locks

to exist for some time until the indexing had completed. This is a not a problem when you first create the “collection” (tables in MySQL) because there are only a few (or no) rows, but creating indexes later will cause problems.

Previously in MySQL we would delete rows by using a wide ranging WHERE clause, for example to delete rows by date range or server ID. Now in MongoDB we have to loop through all the rows and delete them individually. This is slower, but it prevents the locking issue.

Corruption

In MySQL if a database (more likely a few tables) become corrupt, you can repair them individually. In MongoDB, you have to repair on a database level. There is a command to do this but it reads all the data and re-writes it to a new set of files. This means all data is checked and means you will probably have some disk space freed up as files are compacted but it also means the entire database is locked and blocked during the time it takes. With our database being around 60GB, this operation takes several hours.

Corruption will only really occur if you kill the database process whilst it is in the middle of an operation.

Performance

Our reasons for moving to MongoDB were not performance, however it has turned out that in many cases, query times are significantly faster than with MySQL. This is because MongoDB stores as much data in RAM as possible and so it becomes as fast as using something like memcached for the cached data. Even non-cached data is very fast.

We don’t have any precise numbers but in some cases are seeing cached query times around 7ms and non-cached around 50-200ms, depending on the query. Indexes help speed up queries in many cases but where our data is write intensive, indexes can slow things down.

Having a native C PHP module also helps with performance and means that all interactions are optimised at the code level. Other drivers are available for Python, Java, Ruby, C++ and Perl.

Community / commercial support

MongoDB is open source but is developed by a New York company, 10gen. This is useful because we can be sure that development will continue and bugs fixed. Indeed, the mailing list has been a very useful source of help for us during the migration. The documentation is good but some things are still unclear or not documented and being able to get a response from the mailing list from the developers within hours is very helpful.

MongoDB is quite a new project compared to the likes of MySQL and so there are fewer experienced people in the community. As such, we have also taken up a support contract with 10gen for guaranteed 24/7 phone & e-mail coverage so that should we have a problem, we will be able to get help quickly.

Test, and choose what is right for your application

The guys at Friendfeed are using MySQL and they have a lot more data than us. However, they use it like a key/value store and have a different access ratio. Every application is different. Whilst MySQL is suitable for Friendfeed, we found a better solution. You need to test each one to discover its suitability for your needs.

Indeed, whilst Server Density is now running entirely on MongoDB, our accounts system, invoicing and billing remains on MySQL. MongoDB is non-atomic. This doesn’t matter for our general application code – it’s not critical if a few rows do not get written – however this is not the same for our billing system. We use transactions to ensure everything runs correctly (e.g. we don’t bill customers

twice) and so are still using MySQL InnoDB for that.

Our move to MongoDB has been interesting and we have encountered problems, but nothing that we were unable to work around. Performance has increased, our disk usage has decreased and we are now in a much better position to continue our scaling plans.

from → MongoDB, MySQL, Programming, Server Density, Servers, Technical

58 Responses leave one →

1. 
   2009 July 25
   Tero permalink

   Interesting article. It's always nice to read other people experiences, because that way yourself will also learn something new. I see that your needs are quite different from usual database use where read/write rate is quite different, but it would be nice to see some tests how MongoDB make out searching from bigger dataset.

   Becuase I work mostly with some web frameworks, I will wait if some of them will support non-relational database before I will try out MongoDB or some other.

   Reply
2. 
   2009 July 25
   name permalink

   Mongo??! lol :)

   Reply
3. 
   2009 July 25
   hk permalink

   Any comment on why you ended up choosing MongoDB over CouchDB?

   Reply

   - 
     2009 July 26
     David M permalink

     Mongo has very simple querying, much like SQL. CouchDB requires map/reduce style

queries which is much more complicated to work with. Mongo also has the PHP module which is a big advantage.

Reply

- 

  2009 July 26
  Robin Mehner permalink

  CouchDB does not need an extra PHP module, because it communicates via HTTP and PHP has several possibilities to talk via HTTP (cURL, streams, sockets etc.). Also there are some classes around that wrap the access to CouchDB (like phpillow and some more).

  Don’t want to convince you, as stand on your side with your approach, but the “php module” argument isn’t really one against Couch I think :)

- 

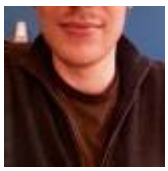
  2009 July 26
  David M permalink

  Indeed, the PHP module argument is not the only reason we decided to go with MongoDB, but it is a plus that is worth mentioning.

- 

  2009 July 27
  Tom Dean permalink

  Unfortunately it’s not possible to achieve the performance of a C/C++ extension through PHP alone. I’m sure for many a library written in PHP is more than adequate, but for anything with serious throughput & volume, a C/C++ extension is the only responsible option (especially when that extension is Boost enhanced).

- 

  2009 July 27
  Jason Watkins permalink

  Actually it’s not so simple. You must measure. For example, there are several .erb template language implementations for ruby, and the fastest is one written in pure ruby, not c++. Why? Because it’s carefully written to avoid allocating intermediate results.

C and C++ do not assure high performance, nor are they always necessary for high performance, particularly for IO heavy workloads. There is no “only responsible option”. You must profile for your workload.

Boost has some excellent features in its’ library, but is frequently bemoaned for poor performance. I find the phrase “boost enhanced” rather silly. The libraries are tools, not an automatic go faster button.

4. 2009 July 26
Thinkapi permalink

Was http://memcachedb.org/ thought out as an option?
Also, http://hadoop.apache.org/hbase/ might have been a good option considering the large amount of data being handled.

Reply

- 
2009 July 26
David M permalink

Both were considered but MongoDB had more advantages. We also didn’t really want to work with map/reduce for querying.

Reply

5. 2009 July 26
nathan permalink

when you outgrow your mongo take a look at SenSage or Vertica. Look up recent performance comparisons between Hadoop & Vertica … complex queries are 10x+ faster w/ Vertica and the same w/ SenSage.

Reply

- 
2009 July 26
David M permalink

Having quickly looked, these both seem like expensive enterprise product unsuitable for a startup.

Reply

- 

  2009 July 26
  Jason Watkins permalink

  Unofficially Vertica is about $25k per TB for production databases (development and testing are free). It is enterprise sales though (bleh) so you may be able to negotiate a better rate.

  Something to bear in mind is column stores such as Vertica compress data very effectively, with real world ratios of 10:1 being typical for transaction data and even higher rates for monitoring/logging data. So your dollar stretches a bit more than it might seem at first. These databases execute against the compressed data directly which also maximizes scanning throughput through the cpu/memory bus.

  If your queries involve large scans compressed column stores are dramatically higher performance than a star schema in a general purpose database. On the other hand, if the majority of your workload is fetching single object's the advantage is reduced to just that from getting more capacity out of your ram.

  I know of at least one startup using Vertica on ec2 and they're quite happy with it.

  I personally have not used it. Just wanted to pass along what I've heard and gleaned from the research literature.

  It'd be great if an open source alternative to Vertica/Teradata/etc existed. Mysql Cluster may get there someday.

6. 

   2009 July 26
   Marton Trencseni permalink

   What kind of replication are you looking for? How many nodes are you replicating to? I'm wondering, would Keyspace be good for you?

   http://scalien.com/keyspace

   Reply

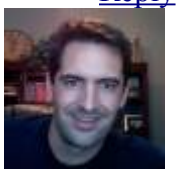
   - 

     2009 July 26
     David M permalink

     Keyspace was released after we'd complete most of the work for migration – I

remember seeing it on HN. MongoDB is a more mature project it seems, even though both are relatively new. There is also a PHP module for MongoDB which is a big advantage for us.

Reply

7. 
2009 July 26
Jim permalink

Excellent article! We have been using MongoDB with Ruby since February and have also been very pleased.

Since you are not using 1 database per customer, how are you segregating your data?

Do you add a collection per customer for all of your customer specific collections?
Did you add a “customer_id” field to your collections?
Other?

Reply

- 
2009 July 26
David M permalink

We add a customer ID prefix to each collection that belongs to them.

Reply

- 
2009 September 12
Chris permalink

Great write-up! Thanks for sharing this information.

Are there any particular reasons why y’all chose to use a collection prefix vs. a ‘customer_id’ field per document?

Trying to figure out a similar situation. I reckon the only downside to using a collection prefix is the (soft?) collection limit per database.

8. 
2009 July 26
Tayssir John Gabbour permalink

Is there any reason MongoDB can't collapse identical field names into a reference to a single string in shared memory? This is a very common optimization, so you're not driven to illegible 2-3 character names.

I suppose an application-level workaround is to define something like what tinyurl.com does — a scheme which replaces big tablenames with smaller ones…

Reply

9. 

2009 July 26
Ross permalink

Wondering why you didn't cosider staying relational w/ PostgreSQL. If you're not going to go map/reduce w/ the queries, key/value stores seem a poor fit to me.

Reply

- 

  2009 July 26
  David M permalink

  MongoDB isn't a key/value store; it's a document store – exactly what we need.

  Have a read through the post above and you'll see the reasons why we switched.

  Reply

10. 

2009 July 27
James permalink

Hi

Thanks for the overview, three questions

* Locks and blocks : can you expand on the impact this might have for client applications – for instance a web app feeding data into mongodb ?

* The PHP module argument is pretty moot – for instance with Couchdb you can access the database via standard socket methods as it's HTTP – you don't have to use a PHP wrapper class – they just wrap-up common operations but with the obvious performance drag.
It'd be interesting to compare Couchdb implementations using sockets Vs MongoDB implementations using the Mongo extension.

Obviously a CouchDB php extension similar to the Mongo extension would be a useful way to avoid PHP wrappers around socket handlers.

BTW: “pecl install mongo” installs the extension using pecl

* “Mongo has very simple querying, much like SQL” – aggregated queries, for example, seem use a reduce method: http://php.net/manual/en/mongocollection.group.php
Do you have an example of aggregated SQL-like queries in Mongo ?

Thanks

Reply

- 
  2009 July 27
  David M permalink

  Locking/blocking would mean that no data can be inserted or read and the client application would hang until it times out or the lock/block is removed.

  The PHP module argument is one of many that add to the overall decision to use MongoDB. In theory, connecting via the PHP module should be faster than setting up an HTTP connection in the PHP code itself because the connection setup will be done in native C rather than having to go through another layer (PHP code).

  We don’t use the MongoCollection Group method, just the query method. This was an advantage because we wanted to avoid the complexity of building map/reduce functions to access our data.

  Reply

- 
  2009 July 27
  David M permalink

  And by query method I really mean “find” e.g.

  $checksMemSwapCol->find(array(’sId’ => (int)$serverId, ‘tA’ => array(‘$gte’ => $timeStartMongo, ‘$lte’ => $timeEndMongo), ‘tAG’ => $granularity), array(‘tA’, ‘vU’, ‘vF’))->sort(array(‘tA’ => 1))->limit(-$limit);

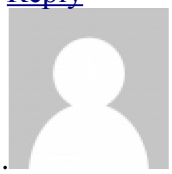
  Reply

11. 
    2009 July 27
    Valentin Kuznetsov permalink

    I’m in process of evaluating MongoDB and I did compare it with CouchDB. I think main difference is support of SQL like queries (which I consider is big plus). It’s easy to understand and more flexible in my mind. I also found that CouchDB has one big disadvantage. View

creation force data indexing. If you must insert a lot of data you'll wait for its access until your view will index all data. It's not required in Mongo. Just to give you idea, 1M records requires more then an hour of indexing time. We did tried bulk injection with parallel clients and ask for data (retrieve 1 doc) each time after bulk insert. That cause a crashes in CouchDB. So I found this instability and not yet solve it CouchDB. What I also interesting in Mongo is bulk injection. Did you tried to insert N docs at a time, where N of the order of 1K or more? What is a best strategy when you need to insert a lot of data? For those who are interesting we're looking for cache solution with DB features (we want queries placed against data in cache). So we must trigger big update from back-end RDMS's.

Reply

12. 

2009 July 28
Joris Verschoor permalink

What's the reason for not choosing tokyo db?

Reply

- 

  2009 July 28
  David M permalink

  None of the client libs I tested worked.

  Reply

  - 

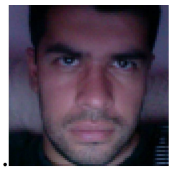
    2010 January 1
    Maurício Linhares permalink

    Tokyo is also horrible for complex querying. But it's a nice key/value store anyway.

    - 

      2010 January 1
      David Mytton permalink

      It's not designed for complex querying, it's a k/v store. You query the key and get the value and that's it.

13. 

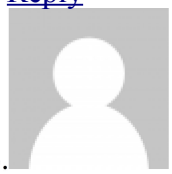
2009 July 29
Ismael Marin permalink

Excellent Article David, thank you for sharing your experience with us.

I hope that in the future we will read another excellent article like this maybe with tips, solutions that you might find after using mongoDB for a long time.

Again thank you and good luck.

Reply

14. 

2009 August 5
Tony Austin permalink

A quite interesting analysis. I'm quite used to the non-relational, document-oriented database paradigm, having worked with Lotus Notes since 1993 (and before that, with more conventional relational DBs).

Just to pick you up on one small point, though. In describing how you saved disk space by restricting the length of field names to a screamingly small 2 to 3 characters, you gave the example of saving 140 MB of disk space for field names. You called this as "a massive saving" and I would disagree. This amount of disk space is trivial these days, when terabyte drives only cost in the order of 100 dollars or so. If you were saving 140 GB then I might agree, but not when the unit of storage space is MB.

I would seriously question whether saving maybe 10 or 20 dollars worth of disk space is worth the trouble caused by your having to battle with cryptically-short field names, a coding nightmare and even more so when it comes to subsequent code maintenance. I thought that we were well past such penny-pinching in these days of commodity hardware pricing.

Cheers,
Tony Austin
CEO, Asia/Pacific Computer Services

Reply

- 

  2009 August 6
  David M permalink

  The hypothetical saving was 760MB for 10m documents, and that is per collection. Although we would only have that many documents for paying customers, spread over all our free customers the saving is quite large. Although the per GB disk usage cost is

relatively low nowadays, any saving where it can be realistically achieved is worth it, particularly as a startup. It could be the difference between having to upgrade to large disks and not, which can be quite a big saving.

Reply

15. 
2009 August 5
JP permalink

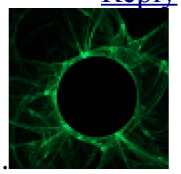
Hi, did you try other mysql engines besides Myisam before moving to Mongodb?

Reply

- 
2009 August 6
David M permalink

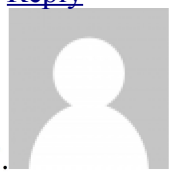
MyISAM was the most suitable for the type of usage we were exeperiencing – many reads and few rights. We used InnoDB (and still do) for the billing and customer systems where we need transactions.

Reply

16. 
2009 August 11
david permalink

awesome i am loving mongodb too :)

Reply

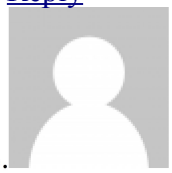
17. 
2009 August 18
Tobias Downer permalink

For a future project, you guys may want to keep an eye on MckoiDDB. It's an open source (GPLv3) distributed DB system – we just made a first public release a couple of days ago. It's also a 'schemaless' key/value database at its roots and we have plans to support some highly structured data models over the base API. The first release contains a hybrid file system and table data model. The idea is, if a developer wants the efficiency of semi-structured data or the expressiveness of structured data models then our API is there to support both.

The system is distributed, transactional and available and handles those things fairly

transparently to the client side developer. Internally it borrows ideas about storing data from modern file systems and the distributed storage parts came from GoogleFS with some of my own ideas thrown in there too :) (it's basically a log-structured file system with a BTree used to represent each instance snapshot). MckoiDDB is in its infancy but you may find the software interesting nonetheless.

Reply

18. 

2009 August 18
Vincent permalink

Thanks for sharing! I'm similarly investigating a better data storage technology than MySQL for our non-relational data in big tables (40mil+ rows). Your information here has proven to be valuable indeed!

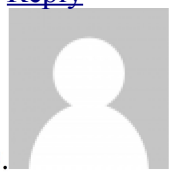
Reply

19. 

2009 September 18
rukeba permalink

You also can look at Redis — http://code.google.com/p/redis/
It is very fast persistent key-value database with built-in net interface.

Reply

20. 

2009 November 17
Bago permalink

It would be cool to understand what kind of data you are storing to MongoDB (all of the server density stuff?) and how you deal with it. How do you denormalize it, how do you aggregate it, how you do real time query it.

Reply

- 

  2009 December 31
  David Mytton permalink

  All the Server Density data is stored in MongoDB – from the list of users added to all the log data. We don't do any de-normalisation and MongoDB allows for very flexible

querying so we have no issues querying against massive datasets – it's all handled by MongoDB.

Reply

21. 

2009 December 31
Matthew permalink

Great article, how do you do relations (because the idea that any database has no relations is ludicrous)? So, for example if you have a Toy Store franchise with many Toy Stores and many Toys within those Toy Stores, is a "document" the individual toy stores and Toys is a hash within a toy store document? And if so, can you index those "subtables"? Or do you makes Toys a different collection, and store Toy Store ids within the Toys documents?

I THINK that's how MongoMapper does it, but that contradicts the non-relational paradigm. If you do it the first way, I'm not sure how many layers down you can go and continue to index, and get good performance. Thoughts?

Reply

- 

  2009 December 31
  David Mytton permalink

  There are no relations, hence the name "non relational database". Of course you're free to create your own associations by including IDs in fields but the database doesn't know anything about them other than they're integer fields.

  I'm not sure what you mean by "toys" behind a hash. MongoDB indexes the content of the field so if it's a hash, you'd have to pass that hash as the key when doing queries or whatever.

  Reply

  - 

    2009 December 31
    Matthew permalink

    Relations, associations, we're just playing word games here. The point was that no database can be without "associations" (whether the DB knows about them or not is a moot point). My question is how do you retrieve information for associated collections without having access to joins? Do you query what you are looking for in one collection and then get it's associated document in another collection?

I guess I'm one of the few people trying to use document-oriented DBs in read-heavy environments because I almost never see much written about querying when it comes to nosql.

- 

  2010 January 1
  David Mytton permalink

  You would need to handle this yourself in your own code. Without joins you'd need to do individual queries one each collection to get the associated data. Where possible you should normalise your data (i.e. duplicate fields) so you can reduce the number of queries needed, but the query cost in MongoDB is very low anyway (assuming you have appropriate indexes of course).

  This is one of the big advantages of MongoDB over something like CouchDB as you can do ad-hoc queries rather than having to construct map reduce syntax (thereby knowing your query in advance).

  NB By "association" I didn't mean in any technical sense.

22. 

    2010 January 1
    ywarnier permalink

    Great article, thanks for sharing, particularly the "unexpected differences".

    Reply

23. 

    2009 September 12
    David M permalink

    Server Density has been developed so it can be deployed in separate installations in addition to our hosted service e.g. if a customer wishes to deploy onto their own servers. The logic for handling the hosted service is therefore outside the core application, which it wouldn't be if we had identifiers in a single collection.

    It also makes it easier to shard based on user.

Evaluating key-value and document stores for short read data

with 16 comments

Designing responsive web interfaces for analyzing short read data requires techniques to rapidly

retrieve and display all details associated with a read. My own work on this has been relying heavily on Berkeley DB key/value databases. For example, an analysis will have key/value stores relating the read to aligned positions in the genome, counts of reads found in a sequencing run, and other associated metadata.

A recent post by Pierre on storing SNPs in CouchDB encouraged me to evaluate my choice of Berkeley DB for storage. My goals were to move to a network accessible store, and to potentially incorporate the advanced query features associated with document oriented databases. I had found myself embedding too much logic about database location and structure into my code while developing with Berkeley DB.

The main factors considered in evaluating the key/value and document stores for my needs were:

- Network accessible
- Python library support
- Data loading time
- Data query time
- File storage space
- Implementation of queries beyond key/value retrieval

Another relevant consideration, which is not as important to my work but might be to yours, is replicating and distributing a database across multiple servers. These are major issues when developing websites with millions of concurrent users; in science applications I am less likely to find that kind of popularity for short read SNP analysis.

Leonard had recently posted a summary of his experience evaluating distributed key stores, which served as an excellent starting point for looking at the different options out there. I decided to do an in-depth evaluation of three stores:

- Tokyo Cabinet, and its network server Tokyo Tyrant, using the pytyrant library.
- CouchDB, using the couchdb-python library.
- MongoDB, using pymongo.

Tokyo Cabinet is a key/value store which received a number of excellent reviews for its speed and reliability. CouchDB and MongoDB are document oriented stores which offer additional query capabilities.

To evaluate the performance of these three stores, frequency counts for 2.8 million unique reads were loaded. Real stores would have additional details on each read, but the general idea is the same: a large number of relatively small documents. Each of the stores was accessed across the network on a remote machine. The 2.8 million reads were loaded, and then a half million records were retrieved from the database. The python scripts are available on github. The below table summarizes the results:

Database	Load time	Retrieval time	File size
Tokyo Cabinet/Tyrant	12 minutes	3 1/2 minutes	24MB
CouchDB	5 1/2 minutes	14 1/2 minutes	236MB
MongoDB	3 minutes	4 minutes	192-960MB

For CouchDB, I initially reported large numbers which were improved dramatically with some small tweaks. With a naive loading strategy, times were in the range of 22 hours with large 6G files. Thanks to tips from Chris and Paul in the comments, the loading script was modified to use bulk loading. With this change, loading times and file sizes are in the range of the other stores and the new times are

reflected in the table. There appear to also be some tweaks that can be made to favor speed over reliability; these tests were done with the standard configuration. The message here is to dig deeper if you find performance issues with CouchDB; small differences in usage can provide huge gains.

Based on the loading tests, I decided to investigate MongoDB and Tokyo Cabinet/Tyrant further. CouchDB loading speeds were improved dramatically by the changes mentioned above; however, retrieval speeds are about 3 times slower. Fetching single records from the database is the most important speed consideration for my work since it happens more frequently than loading, and reflects in the responsiveness of the web front end accessing the database. It is worth investigating whether client code changes can also speed up CouchDB. For Tokyo Cabinet/Tyrant and MongoDB the main performance trade-off was disk space usage for the database files. Tokyo Cabinet loads about 4 times slower, but maintains a much more compact file representation. To better understand how MongoDB stores file, I wrote to the community mailing list and received quick and thoughtful responses on the issue. See the full thread if you are interested in the details; in summary, MongoDB pre-allocated space to improve loading time and this allocation becomes less of an issue as the database size increases.

Looking beyond performance issues, Tokyo Cabinet/Tyrant and MongoDB represent two ends of the storage spectrum. MongoDB is a larger, full featured database providing complex query operations and management of multiple data stores. Tokyo Cabinet and Tyrant provide a lightweight solution for key/value retrieval. Each separate remote Tokyo Cabinet data store requires a Tyrant instance to be running. My work involves generating many different key/value databases for individual short read sequencing projects. To reasonably achieve this with remote Tyrant stores, I would need to develop a server that could start and manage Tyrant instances on demand. Additionally, if my query needs change the key/value paradigm of Tokyo Cabinet would require generating additional key values stores. For instance, we could not readily retrieve reads with a frequency greater than a defined threshold.

In conclusion, both Tokyo Cabinet/Tyrant and MongoDB proved to be excellent solutions for managing the volume and style of data associated with short read experiments. MongoDB provided additional functionality in the form of remote data store management and advanced queries which will be useful for my work; I'll be switching my Berkeley DB stores over to MongoDB and continuing to explore its capabilities. I would welcome hearing about the solutions others have employed for similar storage and query issues.

Possibly related posts: (automatically generated)

-

Written by Brad Chapman

May 10, 2009 at 10:28 am

Posted in storage

Tagged with document-store, keystore, storage

« Finding and displaying short reads clustered in the genome
Python libraries for synthetic biology »

16 Responses

Subscribe to comments with RSS.

1. a good summary. MongoDB looks promising.

would also be nice to see the timings using pytc:
http://github.com/rsms/tc/tree/master
which should be much faster than pytyrant as it is using the c-api, rather than the client protocol. then one could use pytc for fast loading and pytyrant if remote connections are needed.

any thoughts on redis? (http://code.google.com/p/redis/)

brentp

May 10, 2009 at 11:13 am

Reply

2. Great writeup! We're using the ruby bindings for Mongo and our own journey through each of these document stores support the numbers shown here. We're now running http://tweetcongress.org and several other sites off a single MongoDB instance.

   Wynn Netherland

   May 10, 2009 at 11:44 am

   Reply

3. CouchDB's design puts reliability above speed, so for each document saved, it will write out a full index header and fsync to disk. This means that there is no such thing as a fixup phase if your server is rebooted unexpectedly.

   To get high performance writes into CouchDB, it's best to group them into batches of 1k or more documents, and save them using the bulk_docs API. For instance, CouchRest, the Ruby library, has an option to do this automatically.

   Using this bash/curl script I'm able to insert roughly 3k docs per second into CouchDB.

   http://gist.github.com/79279

   The docs my script is inserting are a bit bigger than the key/value pairs you are working with, I'm guessing with your data we'd see a closer to 4k docs/second.

   4k docs/second with 2.8 million docs, will take a little under 12 minutes, which is roughly equal to Tokyo Cabinet in speed. I'll see what I can do about updating your performance script to reflect bulk loading.

   J Chris A

   May 10, 2009 at 11:58 am

   Reply

4. Brad,

   Testing on a Dual Core 2 GHz, 2 GiB RAM Mac Book I can get 3.2 million short read rows inserted into CouchDB in just under 16 minutes. Using Tokyo Cabinet I can do it in 2 minutes. I'm pretty sure that once you make a TC db you can serve it up with tyrant but the port won't install so I can't test this.

   Database sizes were 318 MiB for CouchDB and 152 MiB for Tokyo Cabinet. Not sure why the TC db is so much bigger than the tyrant one though I didn't bother calling optimize when loading it.

   Code is online here:
   http://github.com/davisp/bcbb.git

   The data file I used and preprocessing it are in the comments on the freq_to_couchdb.py script.

22+ hours for CouchDB makes me think you're using version 0.8 which is doing a double fsync for every write to the db. If so, there have been quite a few improvements and options that allow you to choose speed over durability if you so choose.

Paul J. Davis

May 10, 2009 at 1:50 pm

Reply

5. Brent;
pytc would definitely speed things up. This is a good post comparing timings with Cabinet and Tokyo:

http://anyall.org/blog/2009/04/performance-comparison-keyvalue-stores-for-language-model-counts/

For my case I was interested in keeping this distributed if possible; hence the focus on pytyrant.

redis does look good. If I ended up feeling like a key/value store worked better than a document store I likely would have evaluated it as well. This space is definitely blessed with lots of good choices.

Wynn;
Thanks much. I am glad to hear my thoughts were in line with others' experience.

J Chris and Paul;
I appreciate the pointers in the right direction. I figured I was missing something critical; the bulk loading is definitely the way to go. Everything here was done with version 0.9, but without any tuning or modifications at all.

Paul, for the Cabinet/Tyrant size differences, my test here was with on a server with these options:

ttserver test.tcb#opts=ld#bnum=1000000#lcnum=10000

The d gives 'deflate encoding' which likely explains the difference.

Brad Chapman

May 10, 2009 at 7:18 pm

Reply

6. Tokyo Tyrant supports a copying and synchronizing remotely. It's entirely feasible that you could load, copy and clear data sets pretty easily on demand using Tyrant.

I'm interested in your thoughts on Tokyo table support. The table store allows for document-like schema-less storage. Also, you mentioned the need for querying, the Tokyo table store allows for much more complex queries.

Also, as J Chris pointed out about CouchDB, Tokyo also supports bulk read and bulk write. Using ruby-tokyotyrant (native c extension) I can bulk write 2.8m table records in 2m 22secs.

Your closing paragraph seemed light on details on why you chose Mongo, would you care to elaborate on that?

ActsAsFlinn

May 10, 2009 at 10:20 pm

Reply

7. Chris and Paul;
   I had an opportunity to re-run the loading and retrieval this morning using your suggestions for bulk loading, and updated the post to include the numbers doing this the right way. I hope these better reflect your experience using CouchDB.

   Flinn;
   Thanks for the tips on bulk loading for Tyrant. The time difference between Tyrant and MongoDB for loading really didn't both me since loading is an infrequent operation, but that could be very useful for other cases.

   The table support in Toyko C/T does look like it can solve the issue of more complicated queries. The pytyrant bindings currently don't expose any of that functionality, so I didn't have a chance to play with it.

   In terms of my choice of MongoDB, it comes down to providing additional functionality without a significant performance hit. Using Tokyo Cabinet/Tyrant, I'd prefer to not have to write an on-demand server to start various Tyrant instances when needed. Versus CouchDB, my main concern was performance; of course, with the fixes mentioned above the performance of MongoDB and CouchDB do become more comparable.

   Brad Chapman

   May 11, 2009 at 8:04 am

   Reply
8. fwiw MongoDB does have a bulk insert feature — that might make the load times even faster. Not sure if that were used, but looks like it was fast enough anyway. Nice article.

   dwight

   May 11, 2009 at 8:46 am

   Reply
9. Brad,

   Nice write up. I would be interested in your thoughts on a new cloud-based database service we are launching. It is a document-style database service offered via API. Still in stealth mode, but we have an alpha version which is live on AWS. Since you are familiar with these other options your review of this early version of the service would be helpful to us. You can get a test account at http://www.apstrata.com.

   Michael Liss

   May 11, 2009 at 12:01 pm

   Reply
10. You wrote

    My work involves generating many different key/value databases for individual short read sequencing projects.
    To reasonably achieve this with remote Tyrant stores, I would need to develop a server that could start and manage Tyrant instances on demand.

    I'm not sure what you have in mind. Would you not add a "project id" to separate the keys by project (keys look like "project-id:sequence-key") — while using just one Tyrant instance?

    Stephan

    Stephan Wehner

May 18, 2009 at 10:44 pm

Reply

- Stephan;
  Agreed completely; combining identifiers in the key does work and is a trick I'd used in the past with Berkeley DB databases. My impetus to explore document stores was to try and remove this type of logic from my code. In my week or so of using MongoDB since this writeup, I've found the abstraction available with database, collection and document levels very useful in doing this.

  Brad

  Brad Chapman

  May 19, 2009 at 7:04 am

  Reply

-

11. Excellent writeup. I would suggest you check out scalaris too.

    John Laker

    May 19, 2009 at 5:52 am

    Reply

12. [...] are several good blog posts around that go into more detail for each [...]

    Choosing a non-relational database; why we migrated from MySQL to MongoDB « Boxed Ice Blog

    July 25, 2009 at 8:53 am

    Reply

13. [...] is built for speed. Anything that would slow it down (aka transactions) have been left on the chopping block. Instead [...]

    » What If A Key/Value Store Mated With A Relational Database System? endo – explosion kiss

    August 27, 2009 at 7:36 am

    Reply

14. [...] Evaluating key-value and document stores for short read data [...]

    Shane K Johnson » Blog Archive » How I learned to say 'No' to SQL

    September 30, 2009 at 9:23 am

    Reply

15. [...] is built for speed. Anything that would slow it down (aka transactions) have been left on the chopping block. Instead [...]

    What If A Key/Value Store Mated With A Relational Database System? « Ruby Rider

    November 10, 2009 at 12:54 am

16.