Working with Big (lots) Data and Pentaho – Extreme Performance

dankeeley.wordpress.com |

OK, firstly, I'm not talking *proper* BigData here. This is not Hadoop, or even an analytical database. (Lets not get into whether an analytical database counts as bigdata though!). And it's certainly not NoSQL. Disk space we're looking at 100's of gigabytes, not terabytes. Yet this project involves more data than the Hadoop projects I've done.

So tens of billions of records. Records that must be processed in a limited environment in extremely tight time windows. And yes; I'm storing all of that in MySQL!

Hey, wake up, yes, I did say billions of records in MySQL, try not to lose consciousness again… (It's not the first time I've had billions of rows in MySQL either – Yet I know some of you will guffaw at the idea)

In fact, in this project we are moving away from a database cluster, to a single box. The database cluster has 64 nodes and 4TB of RAM. Our single box has 500GB RAM and that was hard fought for after we proved it wasn't going to work with the initial 64GB! Impossible? Yup, that's what I thought. But we did it anyway.

Oh; and just for a laugh, why don't we make the whole thing metadata driven and fully configurable so you never even know which fields will be in a stream. Sure; Lets do that too. No one said this was easy..

Now; how on earth have I managed that? Well firstly this was done with an enormous amount of testing, tuning and general graft. You cannot do something like this without committing a significant amount of time and effort. And it is vital to POC all the way. Prove the concept basically works before you go too far down the tuning route – As tuning is extremely expensive. Anyway we built a solution that works very well for us – your mileage may vary.

I do accept that this is very much at the edge of sanity…

So what did we learn? How did we do this? Some of this is obvious standard stuff. But there are some golden nuggets in here too.

1. Disk usage other than at the start or end of the process is the enemy. Avoid shared infrastructure too.
2. Sorting (which ultimately ends up using disk) is evil. Think extremely hard about what is sorted where.
3. Minimise the work you do in MySQL. Tune the living daylights out of any work done there.
4. MyISAM all the way
5. NO INDEXES on large tables. Truncate and reload.
6. RAM is not always your friend. You *can* have too much.

7. Fastest CPUs you can find (Caveats still apply.. Check specs very carefully Intel do some weird things)
8. Partitioning utterly rocks.
9. Test with FULL production loads or more, PDI/java doesn't scale how you might expect (primarily due to garbage collection), in fact it's impossible to predict. This is not a criticism, it is just how it is.
10. In fact, PDI Rocks.
11. Performance tune every single component separately. Because when you put it all together it's very hard to work out where the bottlenecks are. So if you start off knowing they ALL perform you're already ahead of the game.
12. Use munin or some other tool that automates performance stat gathering and visualisation. But not exclusively. Also use top/iostat/sar/vmstat. Obviously use Linux.
13. What works on one box may not work on another. So if you're planning on getting a new box then do it sooner rather than later.
14. Be prepared to ignore emails from your sysadmin about stratospheric load averages <grin>

I plan to follow this up with a further blog going into details of sorting in PDI in the next few days – complete with UDJC code samples. This post is necessary to set the scene and whet appetites!

Looking forward to hearing other similar war stories.

Of course if anyone wants to know more then talk to me at the next Pentaho London Meetup