



Moving Day:

Migrating your Big Data
from A to B

Laura Thomson
laura@mozilla.com

PGCon 2012

Overview



- What is Socorro?
- The problem
- Planning
- Build out and testing
- Troubleshooting
- Moving Day
- Aftermath

Socorro



[Very Large Array](#) at [Socorro, New Mexico, USA](#). Photo taken by Hajor, 08.Aug.2004. Released under cc-by-sa and/or GFDL. Source: http://en.wikipedia.org/wiki/File:USA_NM_VeryLargeArray.02.jpg



Mozilla Crash Reporter

We're Sorry

Firefox had a problem and crashed. We'll try to restore your tabs and windows when it restarts.

To help us diagnose and fix the problem, you can send us a crash report.

☒ Tell Mozilla about this crash so they can fix it

[Details...](#)

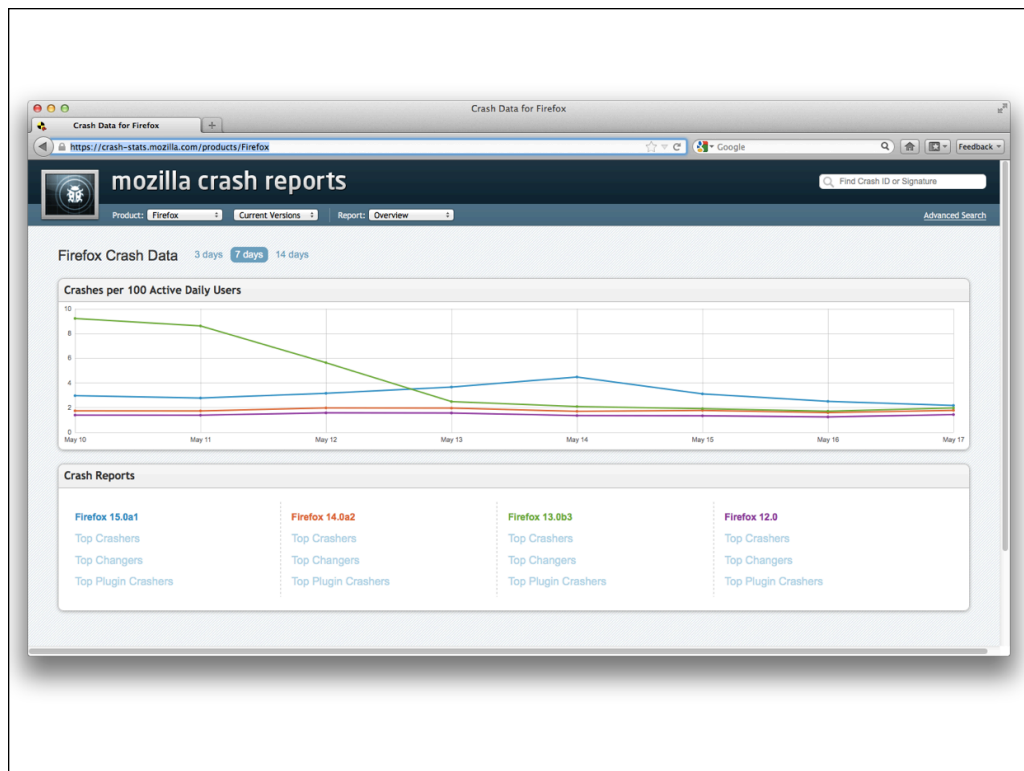
Add a comment (comments are publicly visible)

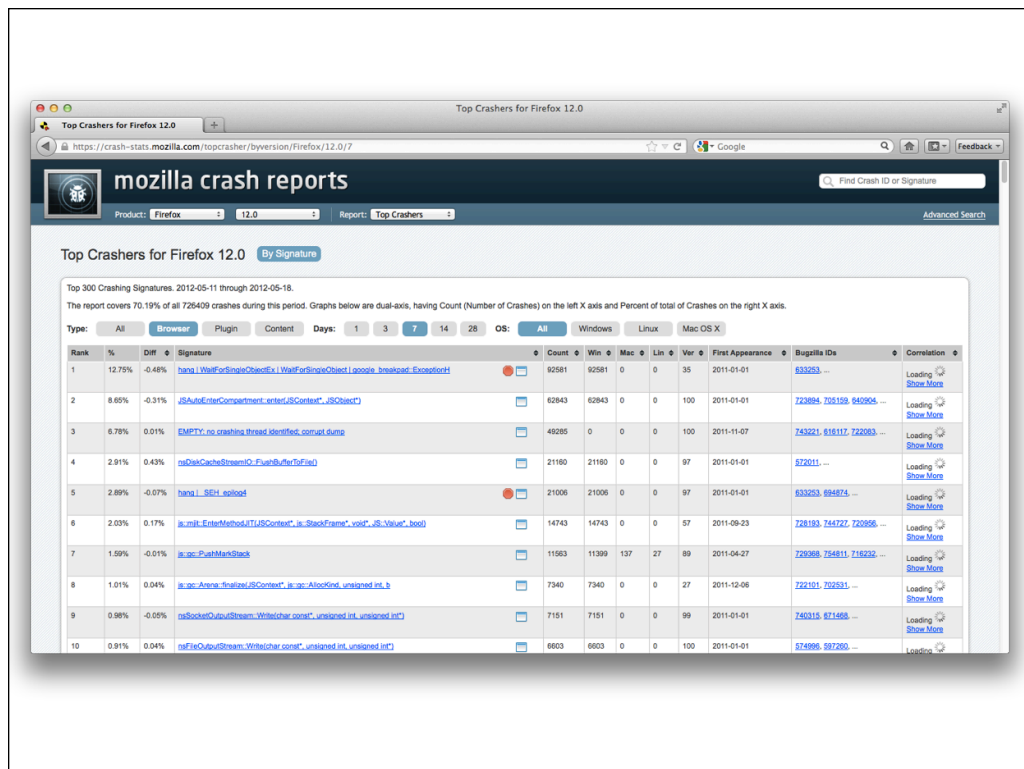
☐ Include the address of the page I was on

☒ Email me when more information is available

Your crash report will be submitted before you quit or restart.

[Quit Firefox](#) [Restart Firefox](#)





Typical use cases

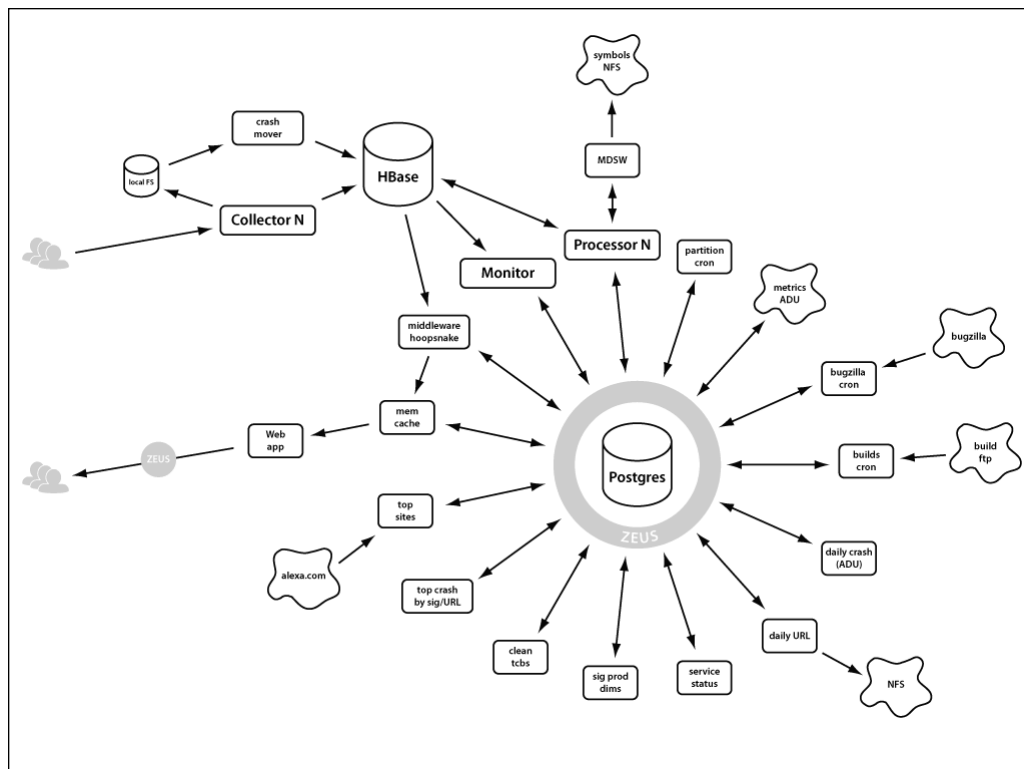


- What are the most common crashes for a product/version/channel?
- What new crashes / regressions do we see emerging? What's the cause of an emergent crash?
- How crashy is one build compared to another?
- What correlations do we see with a particular crash?

What else can we do?



- Does one build have more (null signature) crashes than other builds?
- Analyze differences between Flash versions x and y crashes
- Detect duplicate crashes
- Detect explosive crashes
- Find “frankeninstalls”
- Email victims of a particular crash
- Ad hoc reporting for e.g. tracking down chemspill bugs



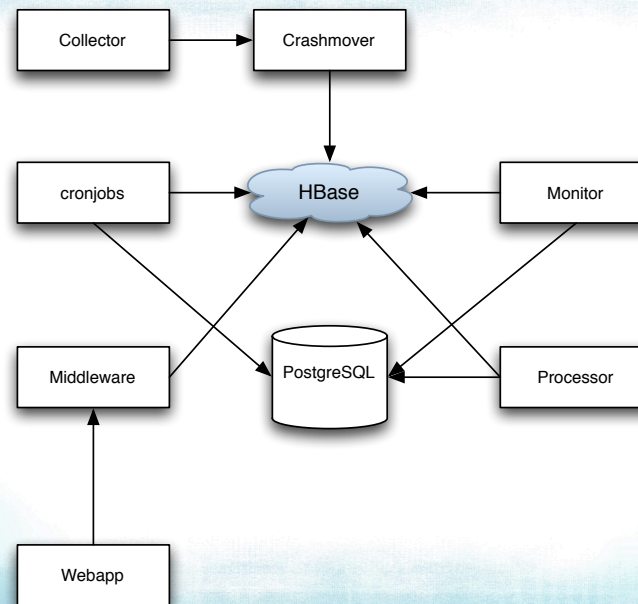


“Socorro has a lot of moving parts”

...

“I prefer to think of them as dancing parts”

Basic architecture (simplified)



Firehose engineering



- At peak we receive 2300 crashes per minute
- 2.5 million per day
- Median crash size 150k, max size 20MB (reject bigger)
 - Android crashes a bit bigger (~200k median)
- ~500GB stored in PostgreSQL - metadata + generated reports
- ~110TB stored in HDFS (3x replication, ~40TB of HBase data)
 - raw reports + processed reports

Implementation scale



- > 120 physical boxes (not cloud)
- ~8 developers + DBAs + sysadmin team + QA + Hadoop ops/analysts
- Deploy approximately weekly but could do continuous if needed

Lifetime of a crash



- Breakpad submits raw crash via POST (metadata json + minidump)
- Collected to disk by collector ([web.py](#) WSGI app)
- Moved to HBase by crashmover
- Noticed in HBase by monitor and assigned for processing

Processing



- Processor spins off minidump stackwalk (MDSW)
- MDSW re-unites raw crash with symbols to generate a stack
- Processor generates a signature and pulls out other data
- Processor writes processed crash back to HBase and metadata to PostgreSQL

Back end processing



Large number of cron jobs, e.g.:

- Calculate aggregates: Top crashers by signature, crashes/100ADU/build
- Process incoming builds from ftp server
- Match known crashes to bugzilla bugs
- Duplicate detection
- Generate extracts (CSV) for further analysis (in CouchDB, f.e.)

Middleware



- All data access through REST API (new)
- Enable other apps against the data platform and allow the core team to rewrite webapp more easily
- In an upcoming version each component will have its own API for status and health checks

Webapp



- Hardest part is sometimes how to visualize the data
- Example: nightly builds, moving to reporting in build time rather than clock time
- Code a bit crufty: rewriting in 2012
- Currently KohanaPHP, will be Django (playdoh)

Other implementation details



- Python 2.6 mostly
- PostgreSQL9.1, stored procs in pgpl/sql
- memcache for the webapp
- Thrift for HBase access
- HBase (CDH3)
- Rolling out ElasticSearch for fulltext indexing of crashes



The problem

The problem



- Approaching capacity (> 85% of storage), causing instability and wanted to store more
- No more power in datacenter and wanted to get out of that datacenter anyway
- Question: How do you move >40TB of data in multiple data stores to a whole new infrastructure in another state...with no downtime?

Complication: Fragility



- Ongoing stability problems with HBase, and when it went down, everything went with it
- Releases were nightmares, requiring manual upgrades of multiple boxes, editing of config files, and manual QA
- Troubleshooting done via remote (awful)
- If we were going to do it over, we were going to do it right.

Analyzing uptime



- Not all parts of a system have the same uptime requirement
- As long as we had zero downtime on data collection, the rest could be offline for a short period (hours, not days).
- This reduces the problem to a tractable one:
 - Collect data to temporary storage (disk) during the migration, and recommence processing once migration complete
- Rewrote crash storage to use a pluggable primary/secondary

Moving data: PostgreSQL



- Theoretically easy!
- Only about 300GB at the time
- Sync from SJC->PHX
- Done in a maintenance window beforehand to reduce downtime on the day, and repeated on migration day
- At this stage we did *not* have replication set up in the old location

Moving data: HBase



- Originally intended to use distcp, an HBase sync utility
- Couldn't use this on a running system, and we couldn't afford the downtime needed (24 hours+)
- Solution: Wrote a dirty copy tool: copy data while running and then use distcp to reach consistency

Planning tools



- Bugzilla for tasks
- Pre-flight checklist and in-flight checklist to track tasks
 - Read Atul Gawande's *The Checklist Manifesto*
- Rollback plan
- Failure scenarios, go/no-go points
- Rehearsals, rehearsals, rehearsals



Build out

Problems with the old system



- Legacy hardware
- Improperly managed code
- Each server was different
- No configuration management
- Shared resources with other webapps
- Vital daemons were started with “nohup ./startDaemon &”
- Insufficient monitoring
- One sysadmin - rest of team and developers had no insight into production
- No automated testing

Configuration Management



- New rule: if it wasn't checked in and managed by Puppet, it wasn't going on the new servers
- No local configuration/installation of anything
- Daemons got init scripts and proper nagios plugins
- Application configuration done centrally in one place
- Staging application matches production

Packages for production



- 3rd party libraries and packages pulled in upstream
- IT doesn't need to know/care how a developer develops. What goes into production is a tested, polished package
- Packages for production are built and tested by Jenkins the same way every time
- Local patches aren't allowed. A patch to production means a patch to the source upstream, a patch to stage and a proper rollout to production
- Every package is fully tested in a staging environment

Load Testing



- Used a small portion (40 nodes) of a 512-node Seamicro cluster
- Simulated real traffic by submitting crashes from the test cluster
- Tested system as a whole, under “real” production load

Troubleshooting



- New data center, new load balancers, new challenges
- Tested and tuned various configurations
- Network misconfigurations reduced performance: discovered and resolved in smoke testing

Migration day



- Flew the team in
- Migration day checklist: <http://tinyurl.com/migrationday>
- Went remarkably smoothly due largely to good co-operation between teams



Aftermath



- Backfilling the data collected during the outage window turned out to be tricky for several reasons:
 - Network flow issues from SJC -> PHX
 - Old submitter in the old datacenter: retroactively upgraded the code to the new multithreaded version to solve that
- Outage in our external ADU data (Vertica failure) the day after made it hard to be sure the data “looked right”

Postmortem



- Postmortem to learn what we did right and wrong
- (Really important to do this, even - especially? - when things go well)

Everything is open (source)



Site: <https://crash-stats.mozilla.com>

Fork: <https://github.com/mozilla/socorro>

Read/file/fix bugs: <https://bugzilla.mozilla.org/>

Docs: <http://www.readthedocs.org/docs/socorro>

Mailing list: <https://lists.mozilla.org/listinfo/tools-socorro>

Join us in IRC: [irc.mozilla.org](https://www.mozilla.org/en-US/contact/#irc) #breakpad and #it

Hiring: <http://mozilla.org/careers>



Questions?