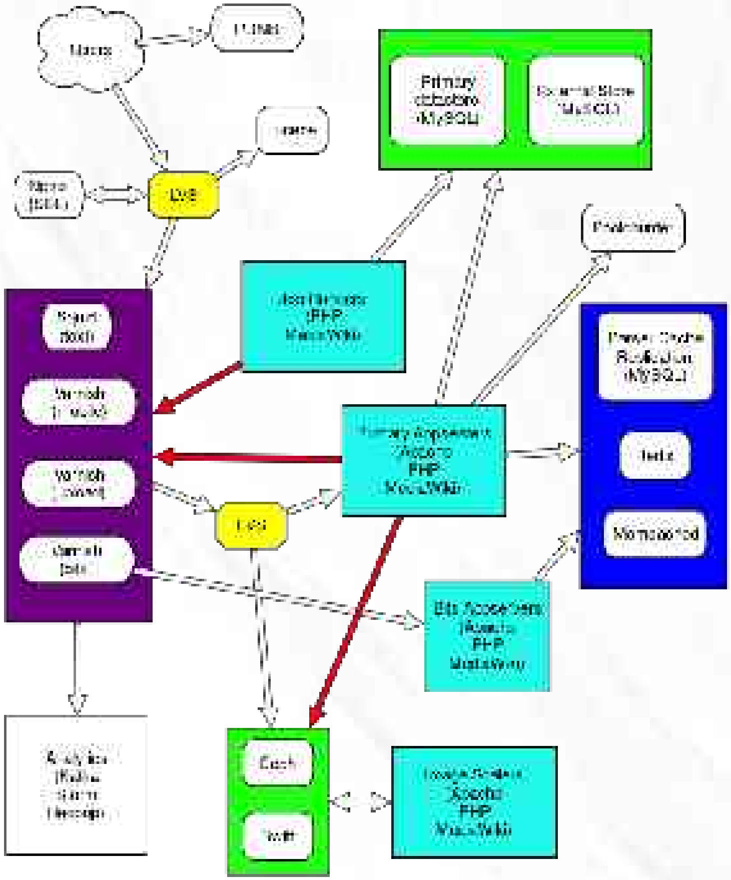
Full-stack performance

Aaron Schulz
5/21/2014

Basic Stack - overview

- **HTML pages:** Varnish → Apaches (MediaWiki)
- **CSS/JS:** Varnish → Apaches (MediaWiki; load.php)
- **Media originals:** Varnish → Swift
- **Media thumbnails:** Varnish → Swift → Apaches (MediaWiki; thumb.php)
- Job runners run various tasks in the background
- Request to api.php routed to different apaches/MySQL
- Apaches and job runners contact various data stores:
  - MySQL, memcached, redis, Swift, Elastic

Basic stack - diagram


Primary datastore (MySQL)
External Store (MySQL)
LVS
Parser Cache Replication (MySQL)
Redis
Memcached
PHP MediaWiki
Primary Appservers Apache PHP MediaWiki
Bits Appservers Apache PHP MediaWiki
Image Scalers Apache PHP MediaWiki
LVS
Ceph
Swift

- File:Wikimedia Server Architecture (simplified).svg
- (c) Ryan Lane

On the edge

- Use geographically spaced CDN nodes
  - Wikimedia uses Varnish (esams,ulsfo,eqiad)
- Use predicable, cacheable, URLs
  - https://en.wikipedia.org/wiki/Hello
  - https://zh.wikipedia.org/zh-hk/萬維網
  - https://zh.wikipedia.org/zh-ch/萬維網
- URLs with special parameters (uncached)
  - http://en.wikipedia.org/wiki/Hello?oldid=6093055559

On the edge

- Common URLs with non-canonical encoding
  - Cached and rewritten in VCL
  - https://gerrit.wikimedia.org/r/#/c/96941/3/modules
- Strange non-cannonical URLs
  - http://en.wikipedia.org/wiki/_Hello
  - Redirects to canonical (low TTL on redirect)

On the edge

- Try to use JavaScript to deliver banners
  - Give the same JS via CDN at let it do the random or geoip bucketing
  - e.g. Central Notice, fundraising
- Do geoip lookups using VCL in the CDN
  - https://git.wikimedia.org/blob/operations%2Fpupp
  - https://git.wikimedia.org/blob/operations%2Fpupp

On the edge

- Package client script modules to avoid RTTs
    - bits.wikimedia.org/en.wikipedia.org/load.ph&lang=en&modules=<...>
- Use base64 data URI for images in CSS
    - img:hover{background:white url(data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAABAAAAAQCAAAAAA6mKC9AAAAGElEQVQYV2N4DwX/oYBhgARgDJjEAAkAAEC99wFuu0VFAAAAAElFTkSuQmCC)...

On the edge

- Avoid cache stampedes from CDN
  - Default in Varnish for duplicate URL requests
  - Called "collapsed forwarding" in Squid
- Avoid excess CPU on any one CDN node
  - Normally use CARP style routing, but VCL was employed to use random routing (Special:CentralAutoLogin)
  - https://git.wikimedia.org/blob/operations%2Fpup

Backend caching - basics

- Avoid cache stampedes for resource
  - CDN de-duplication not always enough
  - Use MediaWiki “PoolCounter” to de-duplicate work (the C daemon is reusable)
  - Using memcached add() can work too :)
- Use memcached to avoid expensive work
  - e.g. Page text, rendered page output
- Increase hit rate by avoiding fragmentation
  - Move customizations to post-processing

Backend caching - stores

- **Memcached**: parser cache, text, random stuff
- **Redis:** user sessions, locking, complex stuff
- **Sharded MySQL (SSDs):** parser cache
- **Swift:** media thumbnails

Database performance - load

- When the DB is needed, try to a slave DB
- Only use the master for anti-dependencies
- LoadBalancer deals with slave lag
  - Periodic slave lag position polling
  - Check slave positions in memcached
  - Avoid slaves with high lag
- Master DBs sharded vertically by project
  - en.wikipedia => s1, commons => s4, ...

Database performance - load

- ChronologyProtector and user expectations
  - Slave master position in user session on edit
  - Next requests wait till the slave gets there
  - Users can see their own edits right after page save without needing the master DB

Database performance - indexes

- Create appropriate indexes
  - Speed up reads and writes
  - Narrow gap locking scope
  - Useless indexes hurt (more so when unique)
- Reuse indexes via redundant WHERE clauses
  - e.g. rev(id,timestamp)=>rc(oldid,timestamp)
- Be careful with many NULLs in an index
  - True for any one value that dominates index
- Periodically run ANALYIZE to update stats

DB performance - transactions

- MediaWiki wraps all queries in a transaction
  - Provides snapshot consistency, avoiding phantom reads and missing foreign keys
  - Exceptions and partitions leave no garbage
  - **Caveat: worse pessimistic locking**

DB performance - transactions

- Keep read transactions short
  - Reduces gap lock contention
  - Avoids unpurged row build up
- Keep write transactions short
  - Reduces slave lag
  - Batch huge queries and move to job queue
  - Move slow method calls **before** contentious queries or **after** the transaction commits

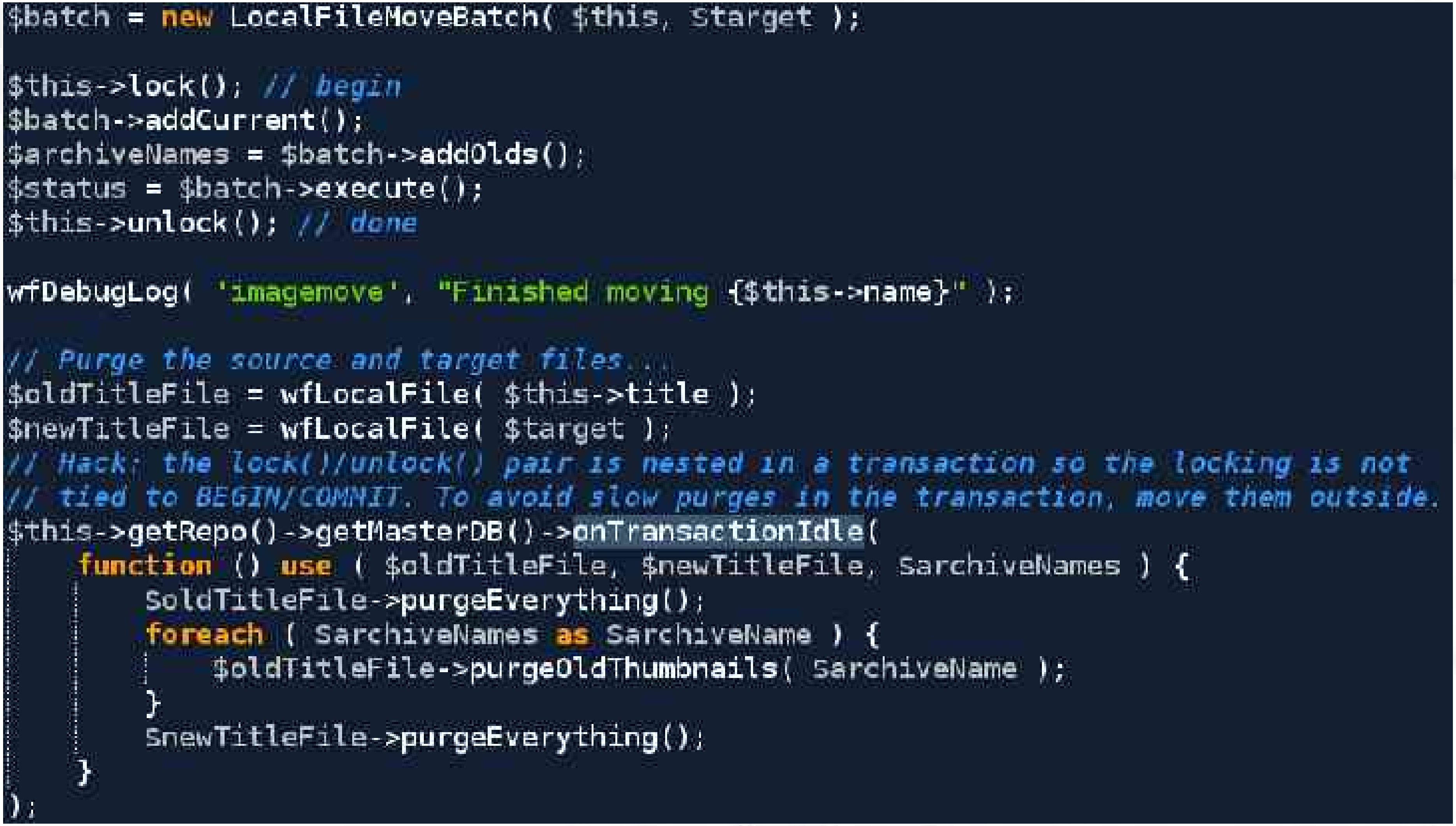
DB performance - transactions

- MediaWiki is a complex monolith, with many extensions subscribed to hooks
- Moving queries around can be difficult
- Solution: $dbw->onTransactionIdle(…)
  - Pass a callback to happen post-commit
  - Useful for contentious updates, cache purges, and slow output dependencies
  - Not 100% atomic anymore

DB performance - transactions

- Solution: onTransactionPreCommitOrIdle(…)
  - Pass in a callback to happen right before COMMIT but after other queries
  - Useful for updating counter fields that need to be atomic with the other changes (or are fast enough that it doesn't matter)
- Passing closures lets us shift responsibility to the DB classes to re-order operations
- *Example: upload new file version, update DB, commi, purge thumbnails*

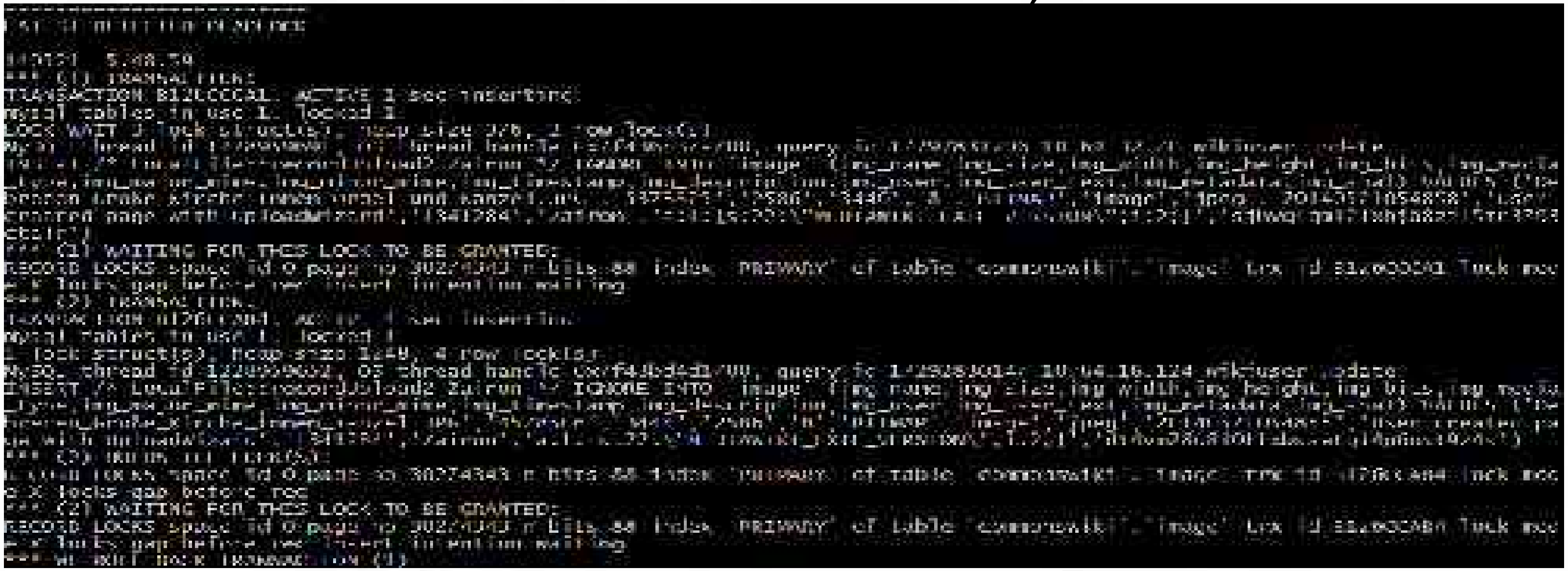
DB performance - transactions



```
$batch = new LocalFileMoveBatch( $this, $target );

$this->lock(); // begin
$batch->addCurrent();
$archiveNames = $batch->addOlds();
$status = $batch->execute();
$this->unlock(); // done

wfDebugLog( 'imagemove', "Finished moving {$this->name}" );

// Purge the source and target files...
$oldTitleFile = wfLocalFile( $this->title );
$newTitleFile = wfLocalFile( $target );
// Hack: the lock()/unlock() pair is nested in a transaction so the locking is not
// tied to BEGIN/COMMIT. To avoid slow purges in the transaction, move them outside.
$this->getRepo()->getMasterDB()->onTransactionIdle(
    function () use ( $oldTitleFile, $newTitleFile, $archiveNames ) {
        $oldTitleFile->purgeEverything();
        foreach ( $archiveNames as $archiveName ) {
            $oldTitleFile->purgeOldThumbnails( $archiveName );
        }
        $newTitleFile->purgeEverything();
    }
);
```

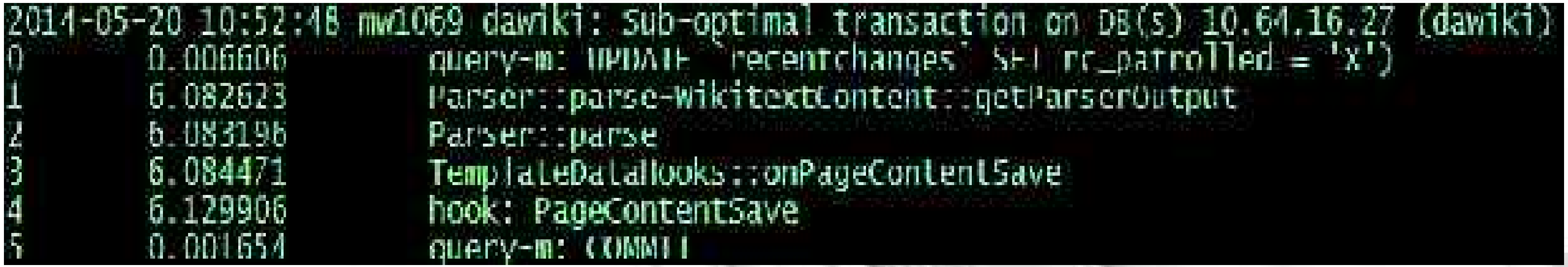
Database performance

- SHOW ENGINE INNODB STATUS :)



```
------------------------
LATEST DETECTED DEADLOCK
------------------------
140521  5:48:59
*** (1) TRANSACTION:
TRANSACTION B12CCCCA1, ACTIVE 1 sec inserting
mysql tables in use 1, locked 1
LOCK WAIT 3 lock struct(s), heap size 376, 1 row lock(s)
MySQL thread id [illegible], OS thread handle [illegible], query id [illegible] 10.64.[illegible] wikiuser update
INSERT /* LocalFile::recordUpload2 Zairon */ IGNORE INTO `image` (img_name,img_size,img_width,img_height,img_bits,img_media
_type,img_major_mime,img_minor_mime,img_timestamp,img_description,img_user,img_user_text,img_metadata,img_sha1) VALUES ('[illegible]
[illegible] kirche innen [illegible] und kanzel.jpg','[illegible]','2586','3440',8,'BITMAP','image','jpeg','20140521054858','User
created page with UploadWizard','1341284','Zairon','[illegible]
[illegible]')
*** (1) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 3027[illegible] n bits 88 index `PRIMARY` of table `commonswiki`.`image` trx id B12CCCCA1 lock mod
e X locks gap before rec insert intention waiting
*** (2) TRANSACTION:
TRANSACTION B12CCCA84, ACTIVE 4 sec inserting
mysql tables in use 1, locked 1
3 lock struct(s), heap size 1248, 4 row lock(s)
MySQL thread id 1228[illegible], OS thread handle 0x7f4[illegible], query id 1[illegible] 10.64.16.124 wikiuser update
INSERT /* LocalFile::recordUpload2 Zairon */ IGNORE INTO `image` (img_name,img_size,img_width,img_height,img_bits,img_media
_type,img_major_mime,img_minor_mime,img_timestamp,img_description,img_user,img_user_text,img_metadata,img_sha1) VALUES ('[illegible]
[illegible]','[illegible]','2586','[illegible]',8,'BITMAP','image','jpeg','20140521054857','User created pa
ge with UploadWizard','1341284','Zairon','[illegible]')
*** (2) HOLDS THE LOCK(S):
RECORD LOCKS space id 0 page no 30274343 n bits 88 index `PRIMARY` of table `commonswiki`.`image` trx id B12CCCA84 lock mod
e X locks gap before rec
*** (2) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 3027[illegible] n bits 88 index `PRIMARY` of table `commonswiki`.`image` trx id B12CCCA84 lock mod
e X locks gap before rec insert intention waiting
*** WE ROLL BACK TRANSACTION (1)
```

- MediaWiki logs of errors and long-held locks



```
2014-05-20 10:52:48 mw1069 dawiki: Sub-optimal transaction on DB(s) 10.64.16.27 (dawiki)
0       0.006606        query-m: UPDATE `recentchanges` SET rc_patrolled = 'X')
1       6.082623        Parser::parse-WikitextContent::getParserOutput
2       6.083196        Parser::parse
3       6.084471        TemplateDataHooks::onPageContentSave
4       6.129906        hook: PageContentSave
5       0.001654        query-m: COMMIT
```

Bulk text storage performance

- **ExternalStore (ES):** sharded MySQL :)
- Revision metadata rows in the primary DB
- `revision` rows => `text` rows => ES URI
- Each cluster is a master and ~2 slaves
- As clusters fill up we add new ones
- *Remark: older clusters have the oldest text...*
- *Maybe use Cassandra instead?*

File storage performance

- Distributed object store (OpenStack Swift)
- Files replicated on 3 nodes; all used for GETs
- Only anti-dependencies use X-Newest: true
- Biggest 25 wiki media containers sharded
  - Swift uses SQLite3 for containers :/
- Purging old thumbnails of files can involve dozens on files...use curl_multi()

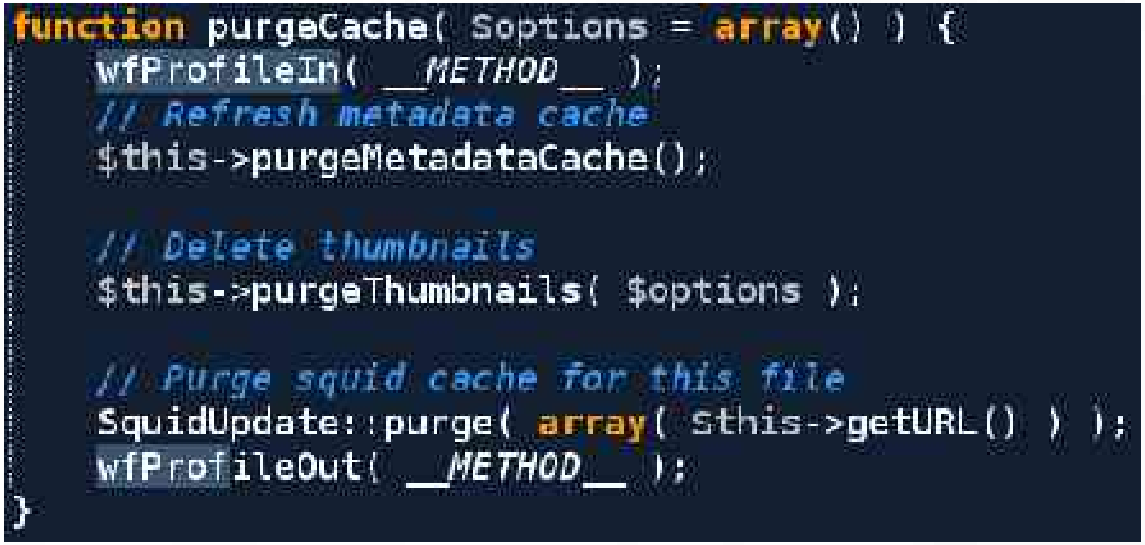
Search performance

- Uses a cluster of ElasticSearch nodes
    - Still migrating from ad-hoc Lucene cluster
- 3 total replicates per index shard
- Mostly per project (e.g. en.wikipedia) indexes
- 1-20 shards per-index (en.wikipedia has 20)

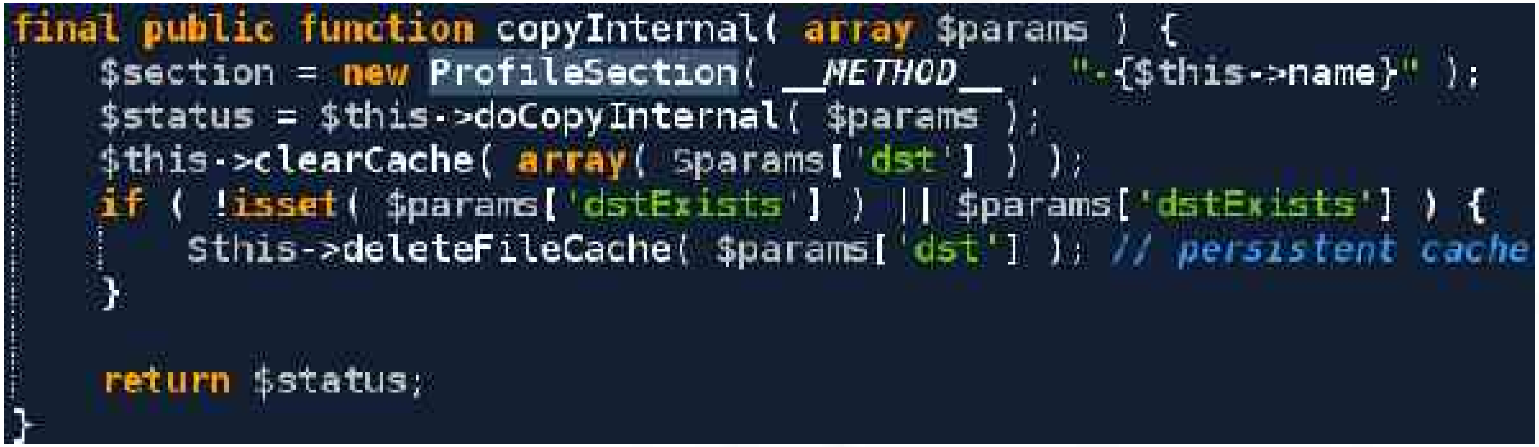
Task queuing

- Job queue is sharded over 2 redis boxes
- Uses LUA commands to Redis
  - Gives atomicity; reduces round trips
  - Supports retries, delays, de-duplication
- Job runner itself is an ugly bash script :)
- Jobs include “slow” tasks like:
  - Varnish, Elastic, HTML, and usage tracking updates when a template changes, emails, mass messages, video transcodes, ...

Profiling – Not Invented Here :)



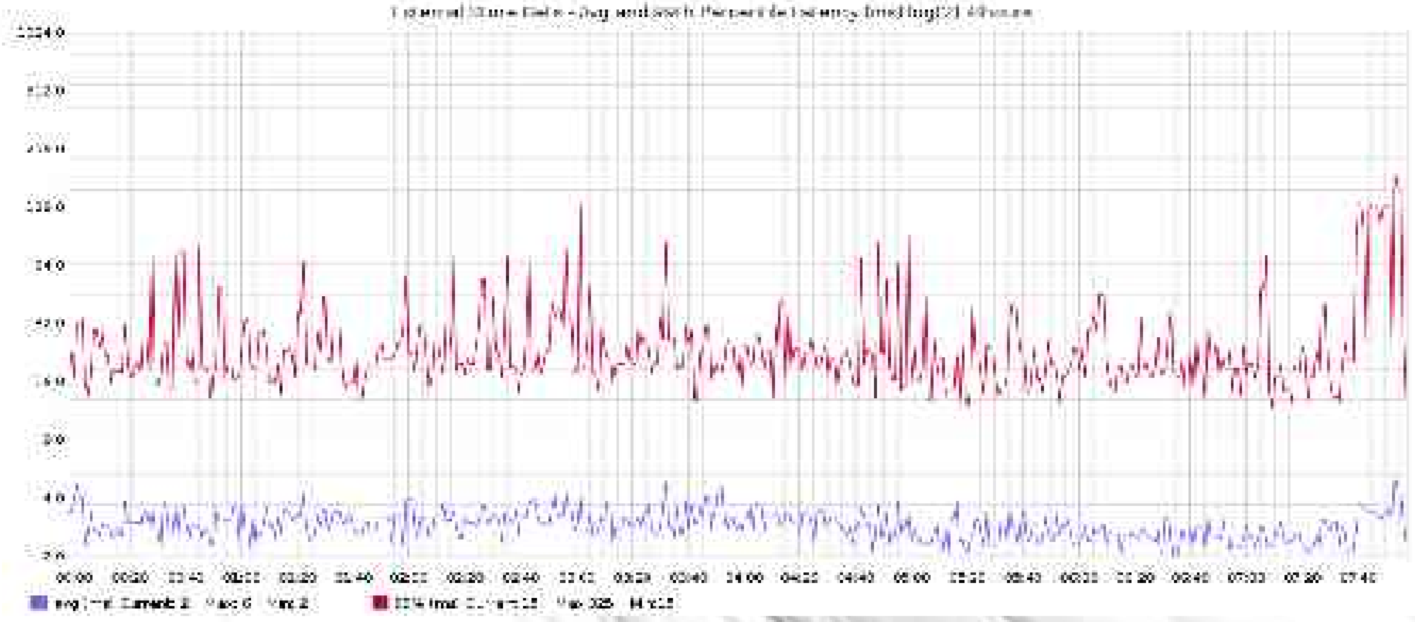
```
function purgeCache( $options = array() ) {
    wfProfileIn( __METHOD__ );
    // Refresh metadata cache
    $this->purgeMetadataCache();

    // Delete thumbnails
    $this->purgeThumbnails( $options );

    // Purge squid cache for this file
    SquidUpdate::purge( array( $this->getURL() ) );
    wfProfileOut( __METHOD__ );
}
```



```
final public function copyInternal( array $params ) {
    $section = new ProfileSection( __METHOD__ . "-{$this->name}" );
    $status = $this->doCopyInternal( $params );
    $this->clearCache( array( $params['dst'] ) );
    if ( !isset( $params['dstExists'] ) || $params['dstExists'] ) {
        $this->deleteFileCache( $params['dst'] ); // persistent cache
    }

    return $status;
}
```

Profiling reports

- performance.wikimedia.org/profiler/report
- graphite.wikimedia.org + gdash.wikimedia.org
  - Stores the data points and lets you build graphs using various filters and functions

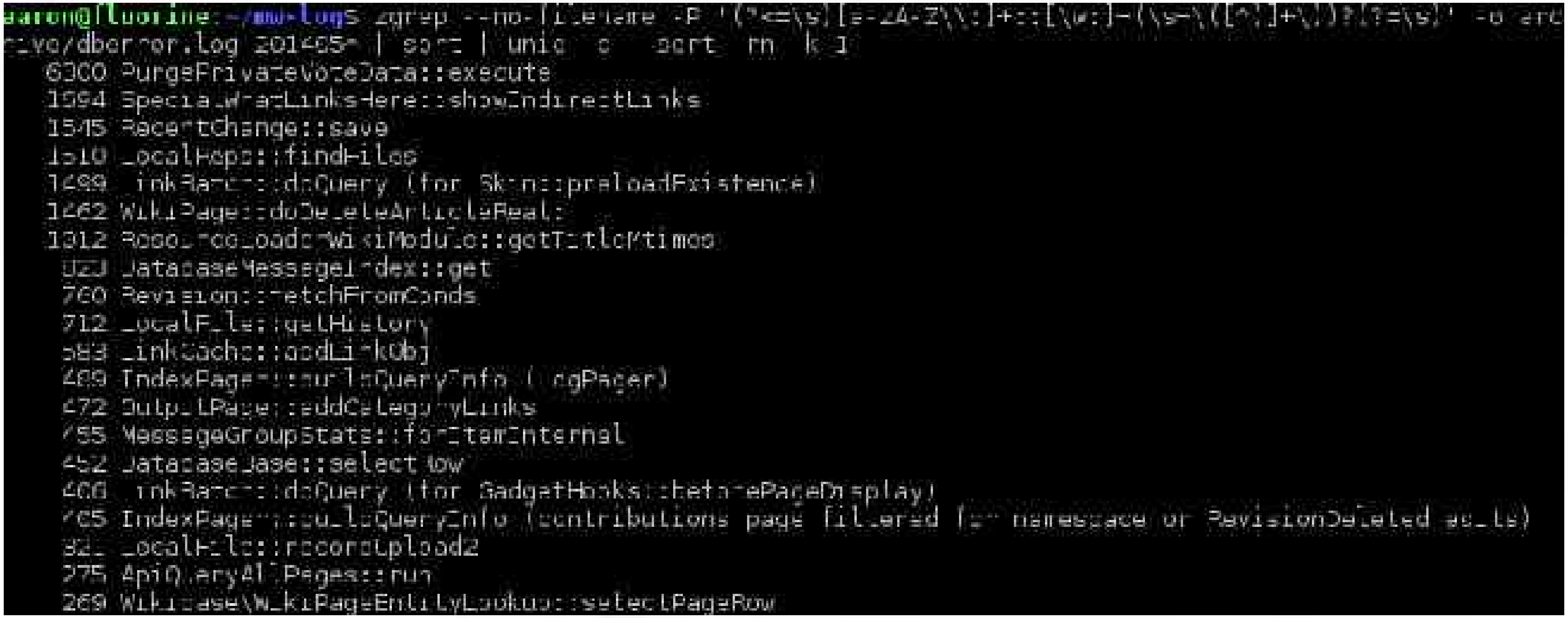


Monitoring & logging

- icinga.wikimedia.org/icinga/
- ganglia.wikimedia.org/latest/
- noc.wikimedia.org/dbtree/
- logstash.wikimedia.org
  - Easier to spot trends than for flat files
- Open source, yay!

Monitoring & logging

- Custom flat file logs on “fluorine” server
    - Useful for AWK/grep + pipe bash-fu



```
aaron@fluorine:~/mw-log$ zgrep --no-filename -P '(?<=\s)[a-zA-Z\\:]+::[\w:]+(\s+\([^)]+\))?(?=\s)' -o archive/dberror.log 201405* | sort | uniq -c | sort -rn -k 1
   6300 PurgePrivateVoteData::execute
   1594 SpecialWhatLinksHere::showIndirectLinks
   1545 RecentChange::save
   1510 LocalRepo::findFiles
   1499 LinkBatch::doQuery (for Skin::preloadExistence)
   1462 WikiPage::doDeleteArticleReal
   1312 ResourceLoaderWikiModule::getTitleMtimes
    823 DatabaseMessageIndex::get
    760 Revision::fetchFromConds
    712 LocalFile::getHistory
    583 LinkCache::addLinkObj
    489 IndexPager::buildQueryInfo (LogPager)
    472 OutputPage::addCategoryLinks
    455 MessageGroupStats::forItemInternal
    452 DatabaseBase::selectRow
    406 LinkBatch::doQuery (for GadgetHooks::beforePageDisplay)
    365 IndexPager::buildQueryInfo (contributions page filtered for namespace or RevisionDeleted edits)
    321 LocalFile::recordUpload2
    275 ApiQueryAllPages::run
    269 Wikibase\WikiPageEntityLookup::selectPageRow
```

General strategy for new code

- Don't worry about micro-optimizations
- Avoid the *obvious* performance pitfalls
  - e.g. try not to scan millions of rows :)
- Non-obvious optimizations need evidence
- Add profiling calls around disk & network I/O
- Deploy, and handle the unforeseen problems
- *Premature optimization can backfire with bugs, cache churn, and extra index overhead*

On the edge - future

- Logged in users bypass cache :/
  - Solution using ESI and JavaScript?
- Pages with many assets slow
  - e.g. http://en.wikipedia.org/wiki/Switzerland
  - Browsers use ~6-8 concurrent connections
  - Not enough; solution using SPDY?
- Varnish uses file nmap with SSDs
  - Eww, hacky...wrap nginx?

Monitoring & logging - future

- Unlike for system failures, there are few performance based SMS alerts
  - People manually check various graphs
  - Regressions that don't totally kill services or saturate the network may persist until users complain or may go unnoticed

Fin