Memcached

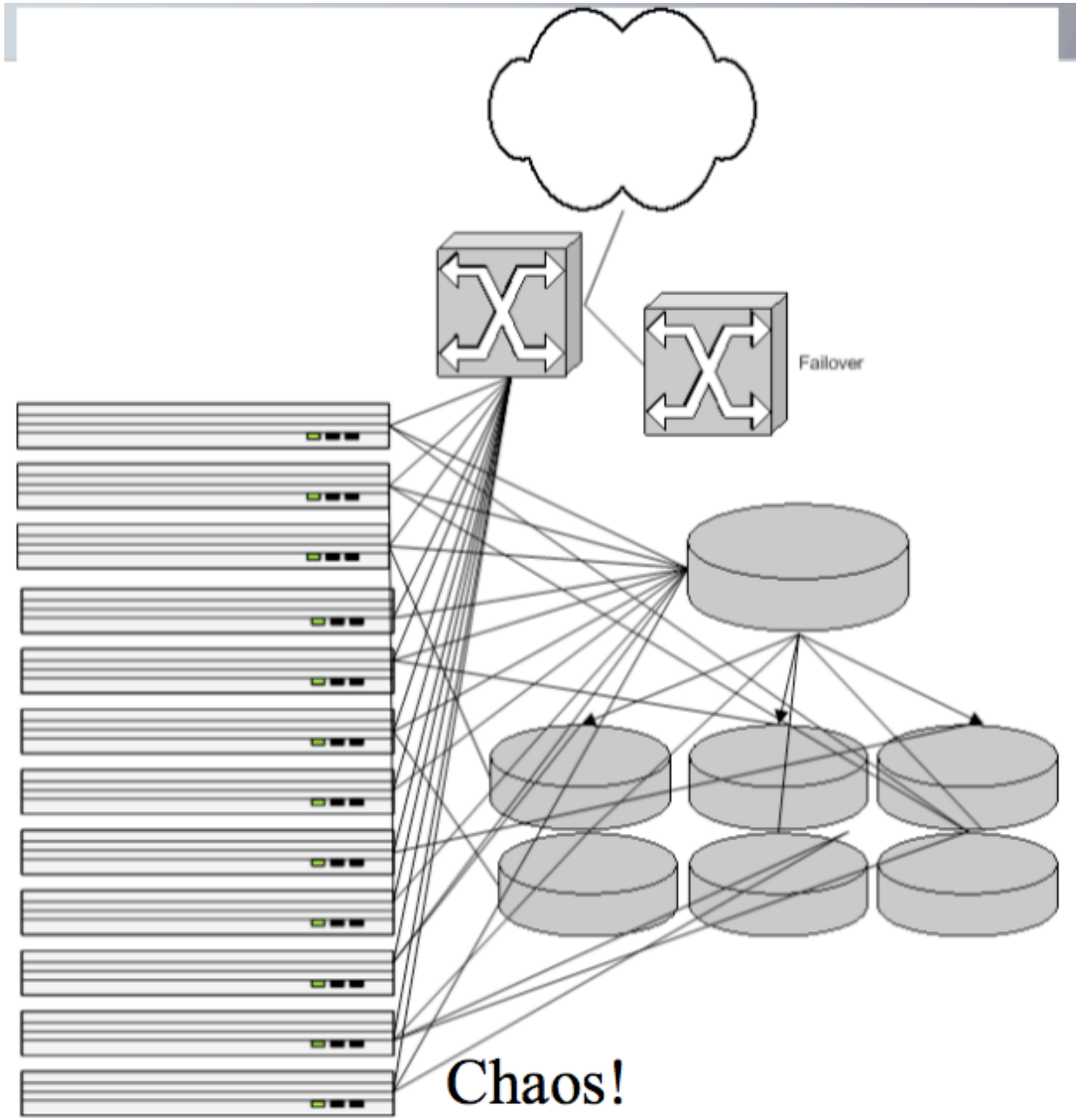
`memcached` is a high-performance, distributed memory object caching system, generic in nature, but intended for use in speeding up dynamic web applications by alleviating database load.

Who?

- Facebook
- Yahoo
- Amazon
- LiveJournal
- Mixi
- ...

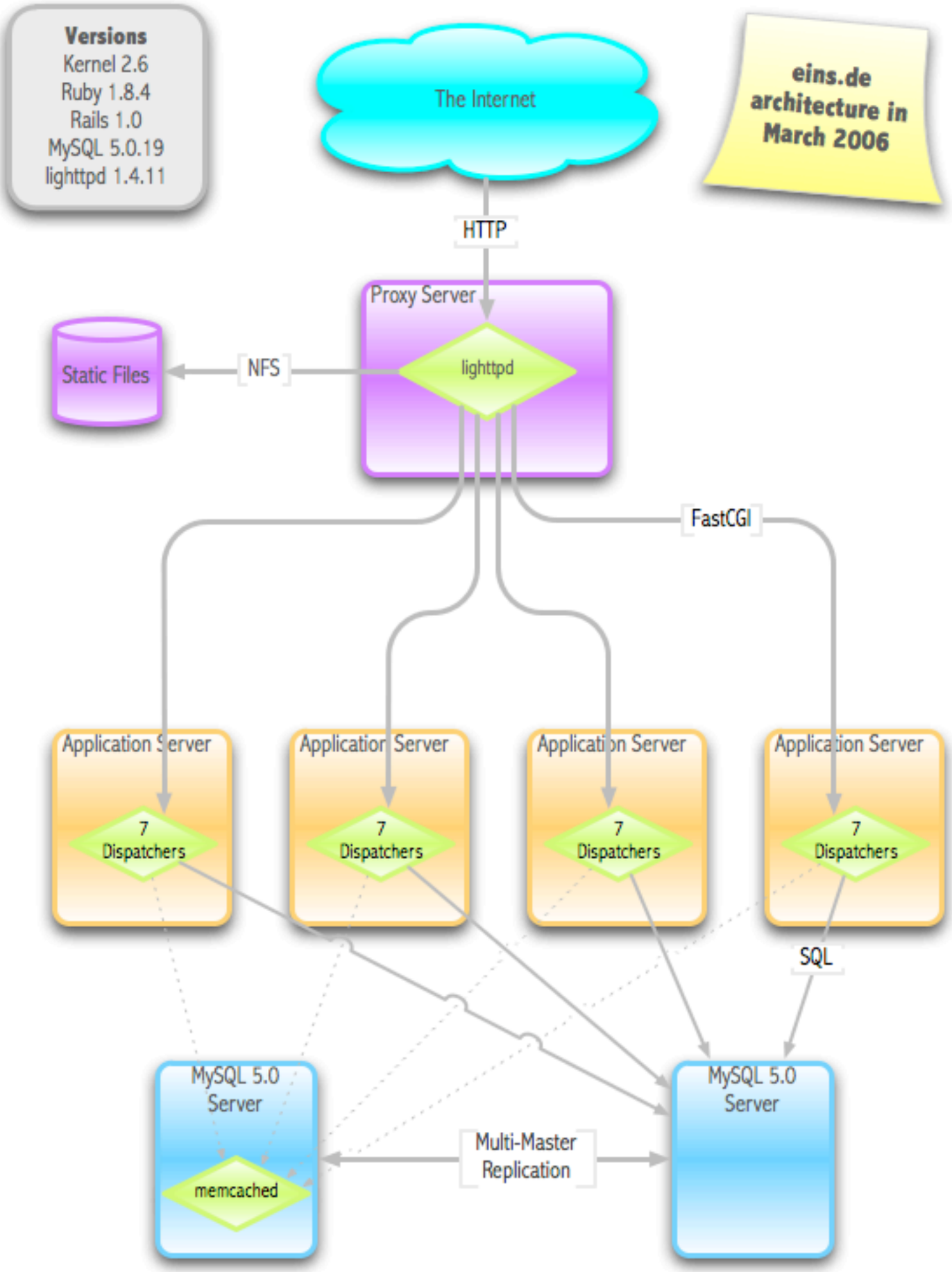
Why?

(aka why would I...)

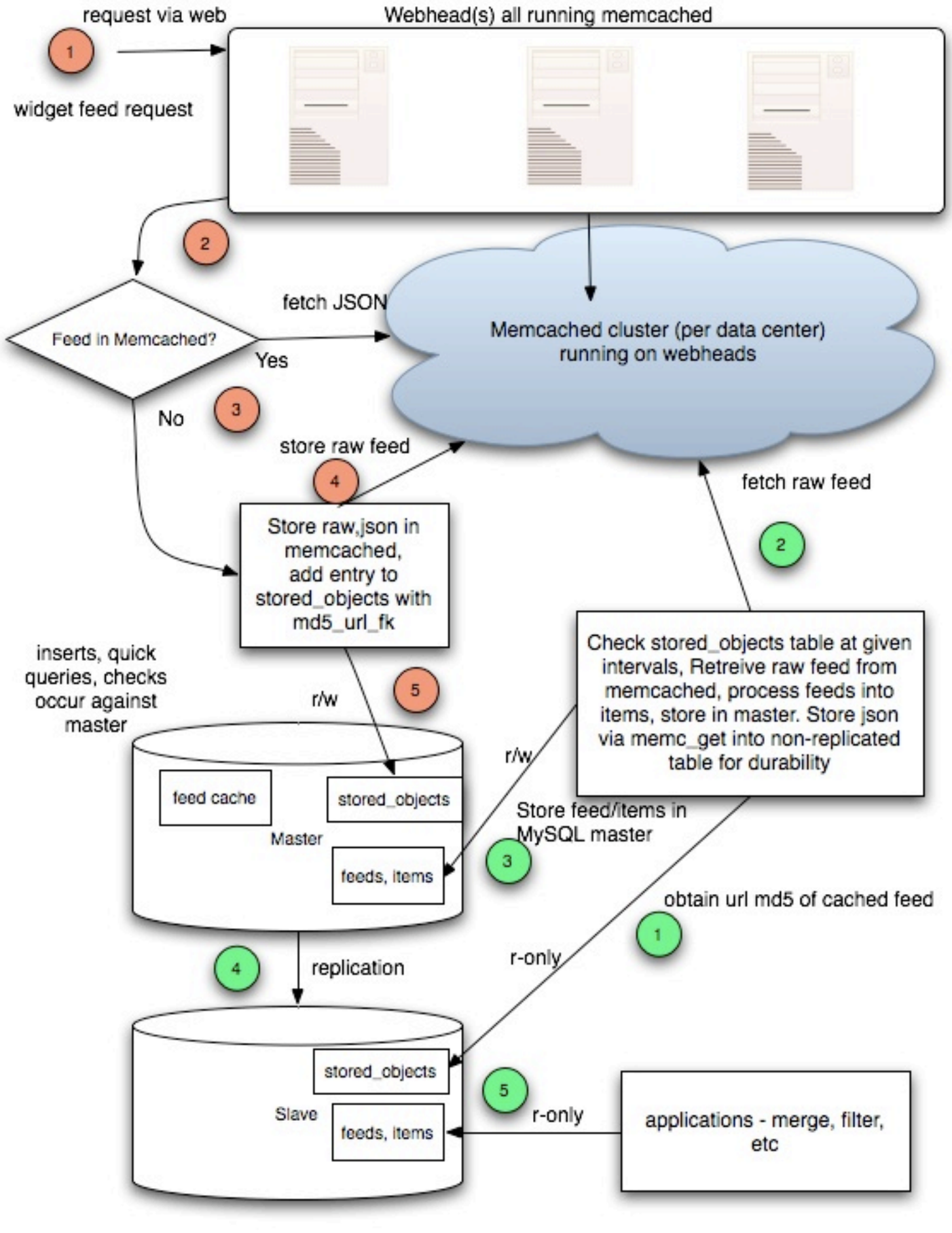

Failover
Chaos!

LiveJournal

- Origin of memcached
- 30G of cache. Terabytes of data
- Writes to DB based on reads from the DB, not cache

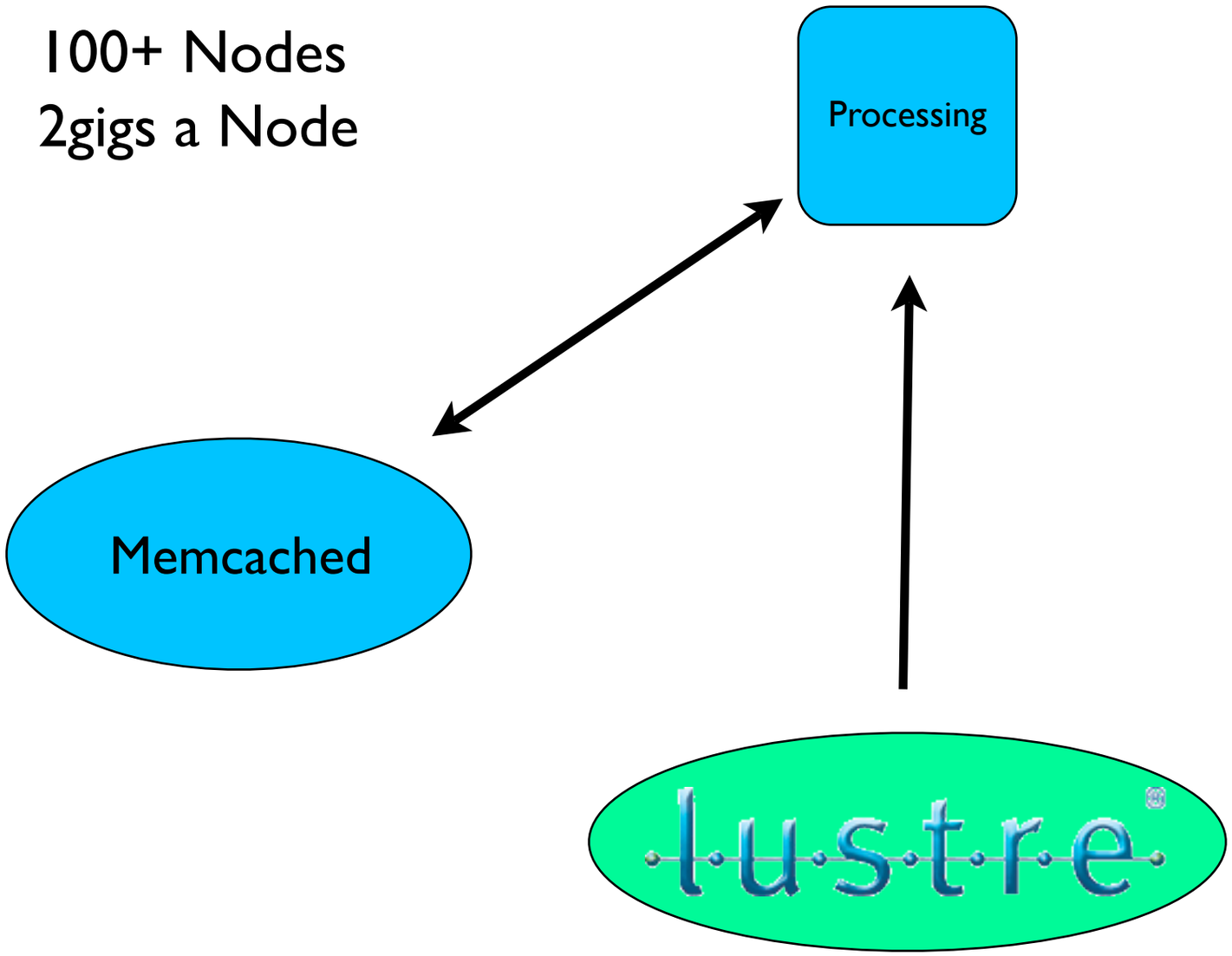


Patrick Lenz
Eins.de
http://poocs.net


request via web
1
widget feed request
Webhead(s) all running memcached
2
Feed in Memcached?
fetch JSON
Yes
No
3
Memcached cluster (per data center)
running on webheads
store raw feed
4
Store raw,json in
memcached,
add entry to
stored_objects with
md5_url_fk
fetch raw feed
2
Check stored_objects table at given
intervals, Retreive raw feed from
memcached, process feeds into
items, store in master. Store json
via memc_get into non-replicated
table for durability
inserts, quick
queries, checks
occur against
master
r/w
5
feed cache
stored_objects
Master
feeds, items
r/w
Store feed/items in
MySQL master
3
obtain url md5 of cached feed
1
4
replication
r-only
stored_objects
Slave
feeds, items
5
r-only
applications - merge, filter,
etc

Grazr

100+ Nodes
2gigs a Node



How

Server

- Slab Allocator
- Libevent based
- Simple Protocol (no xml)
- Server has Internal Hash Table
- Servers know nothing about each other

Clients

- Client hashes Key to Server List (distribution)
- Serializes the Object (server just likes byte arrays)
- Compresses data

So what should I ask?

- How do I dump data?
- How is it redundant? *
- How does it handle failover?*
- How does it authenticate?

Details on the Server?

- set/get/replace
- append/prepend
- increment/decrement
- cas (compare and swap atomic!)
- stats (detail)

Platforms

- FreeBSD and Linux are top tier
- Windows exists
- Solaris (as of 1.2.5)
- OSX Good Support

Application Integration

MySQL Integration

- Most users grab object from database, store object to memcached.
- UDF usage is growing (Patrick's)

Postgres Integration

- pgmemcache()
- Same as MySQL's... but we don't know much about it :)

lighttpd/mod_memcache

- Cache files from disk
- Specify mime/types
- Create Expire Times

Apache (mod_memcached)

- CAS operations exposed
- GET/PUT/DELETE operations
- Still Alpha
- (pandoraport.com!)

Implementation

Server Side

Limits

- Key Size (250 bytes)
- Data Size (under 1 megabyte)
- 32bit/64bit (maximum size of the process)
- Maxbytes (limits item cache, not everything!)

LRU

- Least recently accessed items are up for eviction and can be seen
- One LRU exists per “slab class”
- LRU evictions don't need to be common

Threads

- You might not need them (yet!)
- Great for large instances (16G+)
- Also great for large multiget requests
- Scales okay now. Will get better in the future

Slab Allocator

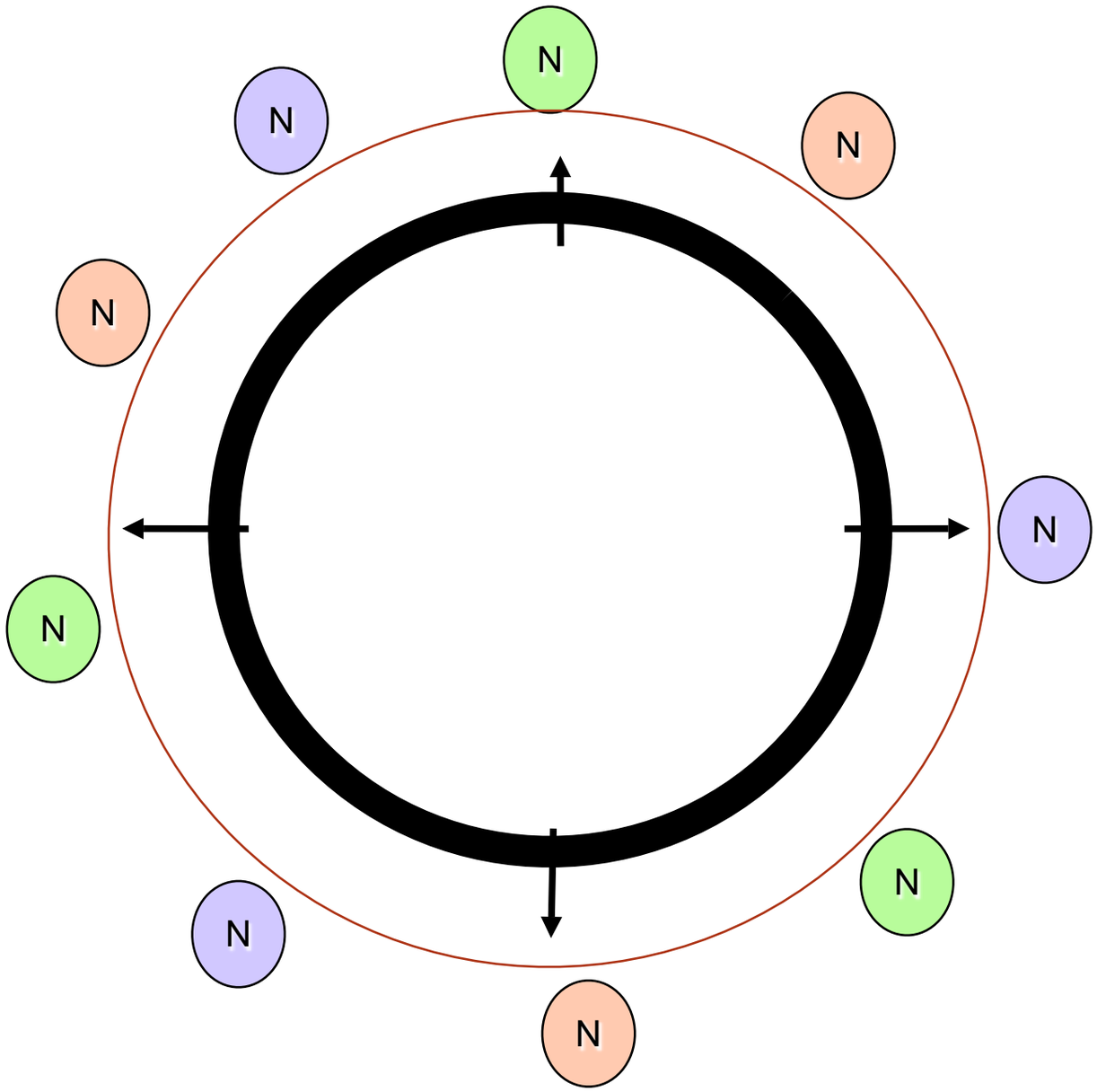
- Memory permanently allocated from OS
- Classes created by chunk size
- Cannot (presently) reassign slab pages
- Carefully use 'stats sizes' to test efficiency

Client

Hashing

- Normal Hashing
- Consistent Hashing

Consistent Hash



Multi-Get vs Single Get

(you want multi... hint, hint...)

Examples

Ruby

```
# Set up client object

require 'memcache'
servers = ['127.0.0.1:43042', '127.0.0.1:43043']
CACHE = MemCache.new(servers, :namespace => 'my_app')
```

```
# Get; fall through to Rails' MySQL load if missing

 key = "recent_posts"
 # Try to get; returns nil on failure
 posts = CACHE.get(key)

unless posts
  # Load from DB
  posts = Post.find(:all, :limit => 10, :order => 'id DESC')
  # Cache the new value, which is automatically serialized
  CACHE.set(key), posts, 60
end
```

```
# Ghetto locking implementation for memcache-client
 # from http://fauna.rubyforge.org/svn/interlock/trunk/lib/interlock/lock.rb

 def lock(key, lock_expiry = 30, retries = 5)
   retries.times do |count|
     # Try to acquire the lock
     response = CACHE.add("lock:#{key}", "Locked by #{Process.pid}", lock_expiry)
     if response == "STORED\r\n"
       # We got it
       begin
         # Yield the current value to the closure
         value = yield(CACHE.get(key))
         # Set the new value returned from the closure CACHE.set(key, value)
         # We're done ('ensure' block will still run)
         return value
       ensure
         # Release the lock
         CACHE.delete("lock:#{key}")
       end
     else
       # Exponentially back off requests if the lock can't be acquired
       sleep((2**count) / 2.0)
     end
   end
   # We waited and waited but our turn never came
   raise MemCacheError, "Couldn't acquire lock for #{key}"
 end
```

Perl

```
sub get_foo_object {
   my $foo_id = int(shift);
   my $obj = FooClass::MemCache->get("foo:$foo_id");
   return $obj if $obj;

   $obj = FooClass::db->selectrow_hashref("SELECT .... FROM foo f, bar b ".
                                          "WHERE ... AND f.fooid=$foo_id");
   FooClass::MemCache->set("foo:$foo_id", $obj);
   return $obj;
}
```

PHP

```
/**
 * Initalize Memcache object and add local server 127.0.0.1 using port 11211
 */
$cache = new Memcache;
$cache->addServer("127.0.0.1",11211);
```

```
/** Look in cache, if not found, set object **/
if (!$user = $cache->get($user_key)) {

  /**
   * Set object with expire of 1 hour and no compression
   */
  $cache->set($user_key,$value,NULL, $expire);
  $user = $value;
}
```

```
/* Get user counter value */
if (!$counter = $cache->get($counter_key)) {
  /* No counter, set to 1 */
  $cache->set($counter_key,1);
} else {
  /* Increment by 1 */
  $cache->increment($counter_key);
}
```

```
/* Print out stats from memcached server */
print_r($cache->getStats());

/* Print out extended stats from memcached server */
print_r($cache->getExtendedStats());

/* Flush memcached (bad idea in production)*/
var_dump($cache->flush());
```

C

libmemcached

- C/C++ (many language wrappers)
- Multiget support
- Async/Sync Modes (including buffered)
- Replicated
- Read Through Cache Support

```
memcached_st *memc;
memcached_return rc;
memc= memcached_create(NULL);
...do stuff...
memcached_free(memc);
```

```
memcached_server_st *servers;
 memcached_st *memc= memcached_create(NULL);
 char servername[]= "0.example.com";
 servers= memcached_server_list_append(NULL, servername, 400, &rc);
 for (x= 0; x < 20; x++)
 {
  char buffer[SMALL_STRING_LEN];
  snprintf(buffer, SMALL_STRING_LEN, "%u.example.com", 400+x);
  servers= memcached_server_list_append(servers, buffer, 401, &rc);
 }
rc= memcached_server_push(memc, servers);
memcached_server_free(servers);
memcached_free(memc);
```

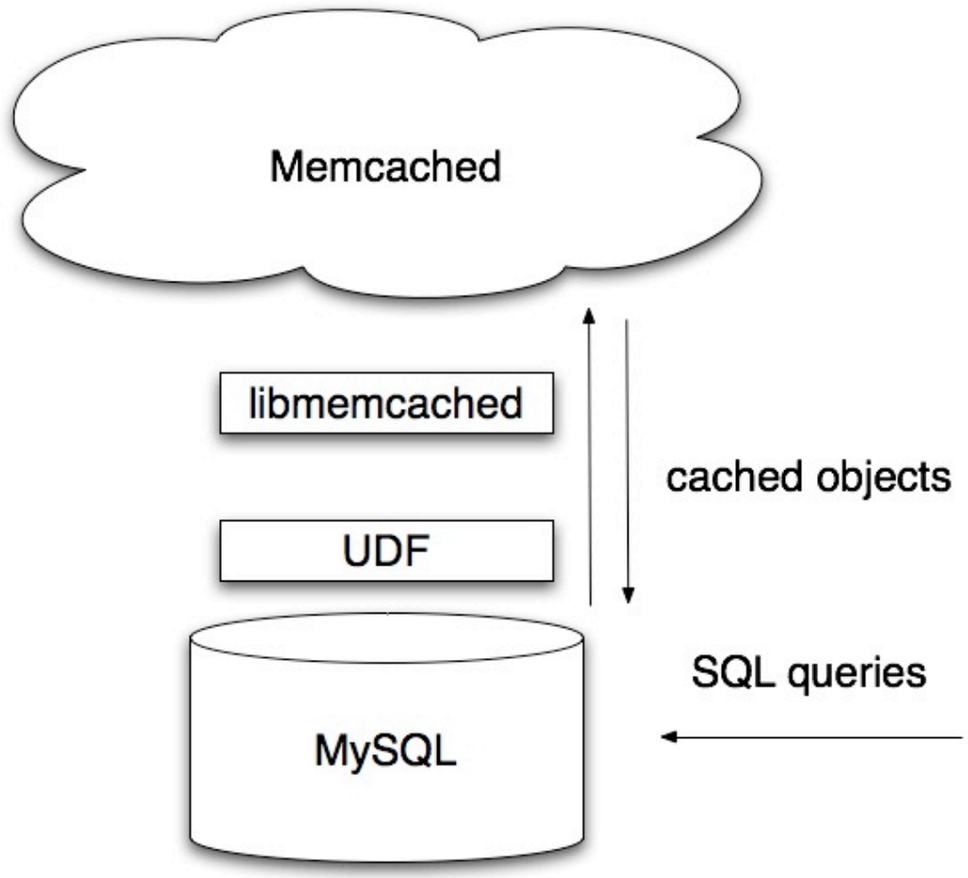
```
char *key= "foo";
char *value;
size_t value_length= 8191;
unsigned int x;
value = (char*)malloc(value_length);

for (x= 0; x < value_length; x++)
  value[x] = (char) (x % 127);
for (x= 0; x < 1; x++)
{
  rc= memcached_set(memc, key, strlen(key),
  value, value_length,
  (time_t)0, (uint32_t)0);
  assert(rc == MEMCACHED_SUCCESS);
}
free(value);
```

```
memcached_return rc;
char *keys[]= {"fudge", "son", "food"};
size_t key_length[]= {5, 3, 4};
uint32_t flags;
char return_key[MEMCACHED_MAX_KEY];
size_t return_key_length;
char *return_value;
size_t return_value_length;

rc= memcached_mget(memc, keys, key_length, 3);

while ((return_value= memcached_fetch(memc, return_key, &return_key_length,
                                      &return_value_length, &flags, &rc)))
{
  free(return_value);
}
```

Memcached Functions for MySQL

Overview...

- Uses UDF API and libmemcached
- Manage memcached Cluster via SQL
- Read through Cache
- Write through Cache


Memcached
libmemcached
UDF
MySQL
cached objects
SQL queries

Installation

- CREATE FUNCTION memc_servers_add RETURNS INT SONAME "libmemcached_functions_mysql.so";
- CREATE FUNCTION memc_set RETURNS INT SONAME "libmemcached_functions_mysql.so";
- CREATE FUNCTION memc_get RETURNS STRING SONAME "libmemcached_functions_mysql.so";

Functions Available

- memc_servers_set();
- memc_servers_behavior_set()
- memc_set()
- memc_get()
- memc_append()
- memc_prepend()

Setting via SELECT

```
select id, url, memc_set(concat('feeds', md5(url)), url) from feeds;

select memc_get(concat('feeds', md5(url))) from feeds;
+-------------------------------------------------+
| memc_get(concat('feeds', md5(url)))             |
+-------------------------------------------------+
| http://feeds.feedburner.com/littlegreenfootballs/Ilds |
| http://del.icio.us/rss/jacomien                 |
| http://del.icio.us/rss/tags/rmj20               |
| http://www.jihadwatch.org/index.rdf             |
| http://www.connotea.org/rss/user/rmj20          |
| http://devblog.grazr.com/?feed=rss2x            |
| http://feeds.feedburner.com/littlegreenfootballs/kyeH |
+-------------------------------------------------+
```

Trigger

```
DROP TRIGGER IF EXISTS feed_insert;
CREATE TRIGGER feed_insert BEFORE INSERT ON feedsFOR EACH ROW
BEGIN   SET @mm= memc_set(concat('feeds:',md5(NEW.url)),
NEW.url);END |
```

```
insert into feeds (url) values ('http://grazr.com/feedlist.xml');

select memc_get(concat('feeds:', md5('http://grazr.com/feedlist.xml')));
+--------------------------------------------------------------------+
memc_get(concat('feeds:', md5('http://grazr.com/feedlist.xml'))) |
+--------------------------------------------------------------------+
http://grazr.com/feedlist.xml                                        |
+--------------------------------------------------------------------+
```

Tools

Protocol

- Telnet'able (see doc/protocol.txt)
- All commands are text based (binary soon)
- Store commands are binary (no escaping)

memcached-tool

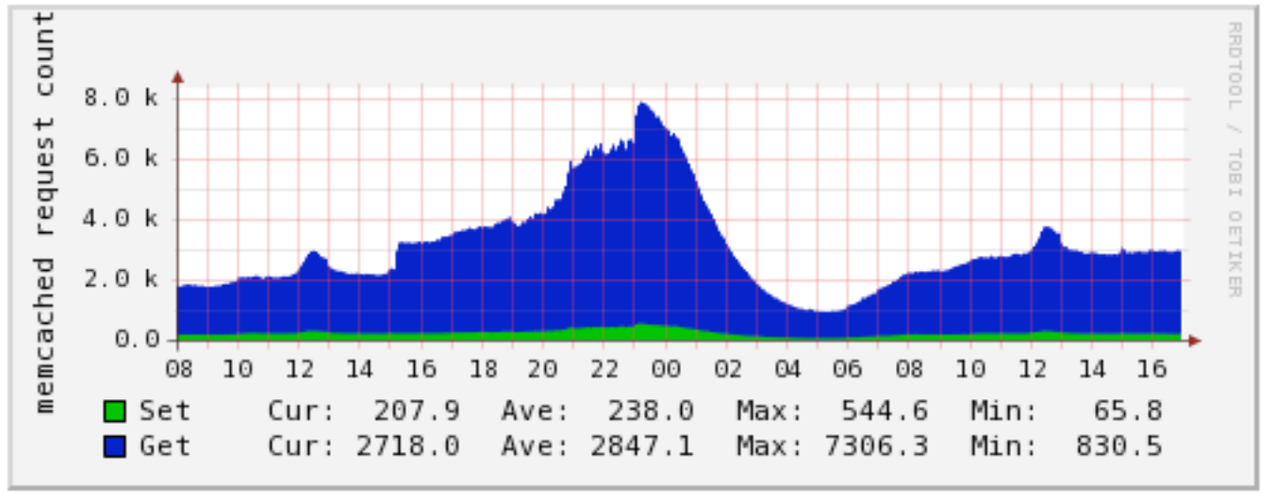
- Example minimal monitoring tool
- Shows details of server stats, slabs
- memcached-tool 10.0.0.1 display
- memcached-tool 10.0.0.1 stats

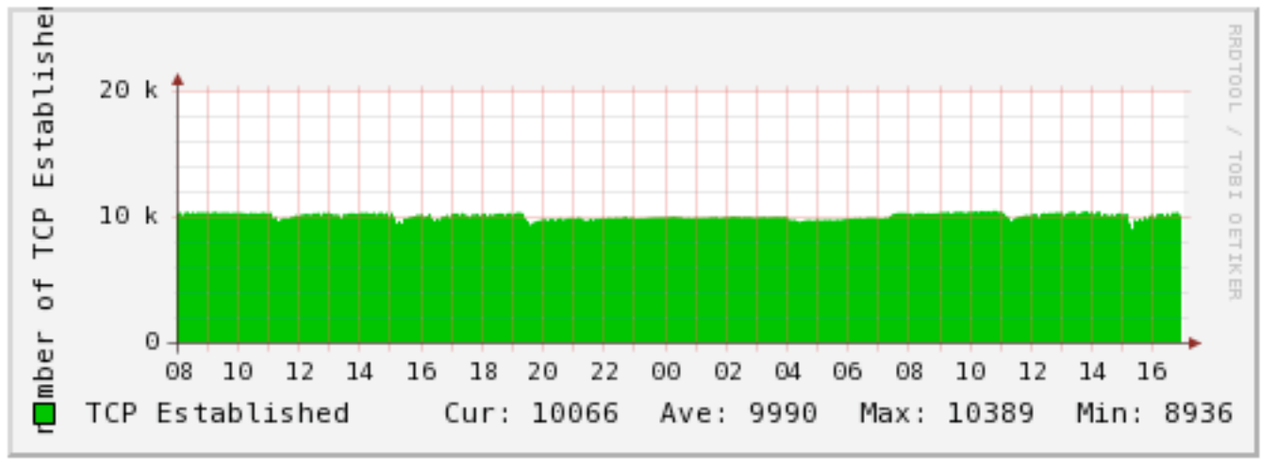
libmemcached Tools

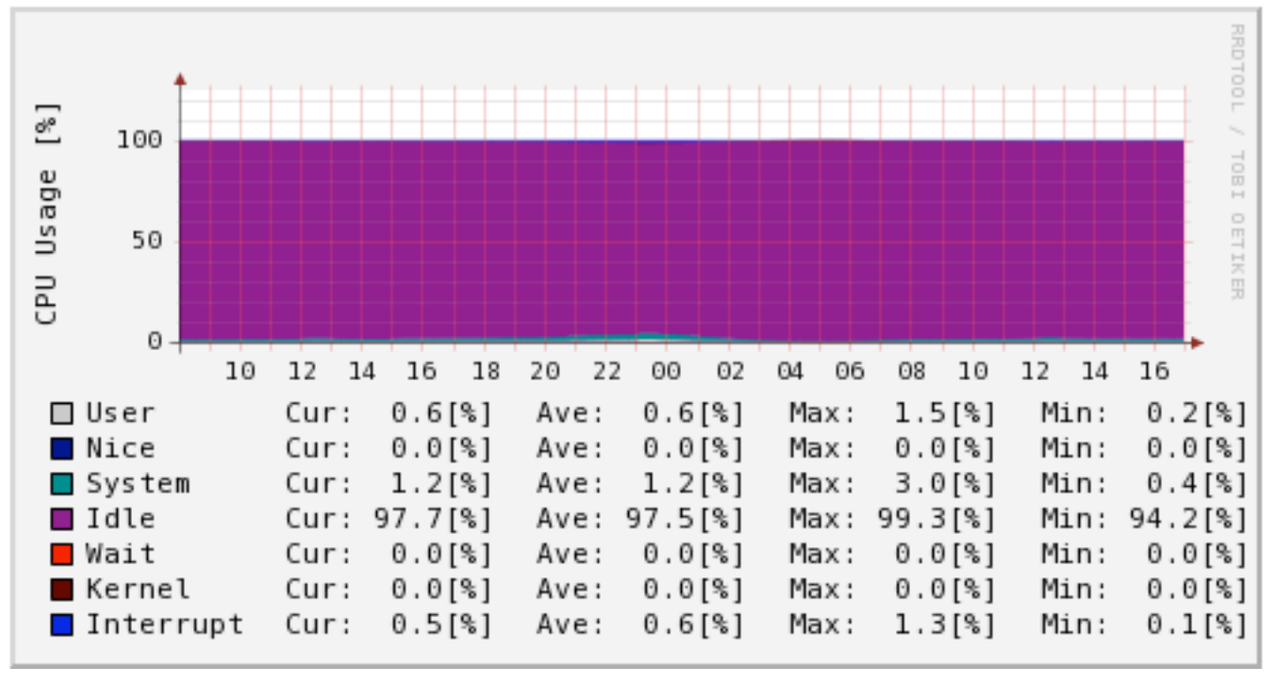
- memcp
- memrm
- memstat
- memslap (hahahhaha)

MRTG

- Plenty of interfaces monitored
- Old technology, been around forever


memcached request count
8.0 k
6.0 k
4.0 k
2.0 k
0.0
08 10 12 14 16 18 20 22 00 02 04 06 08 10 12 14 16
Set Cur: 207.9 Ave: 238.0 Max: 544.6 Min: 65.8
Get Cur: 2718.0 Ave: 2847.1 Max: 7306.3 Min: 830.5
RRDTOOL / TOBI OETIKER

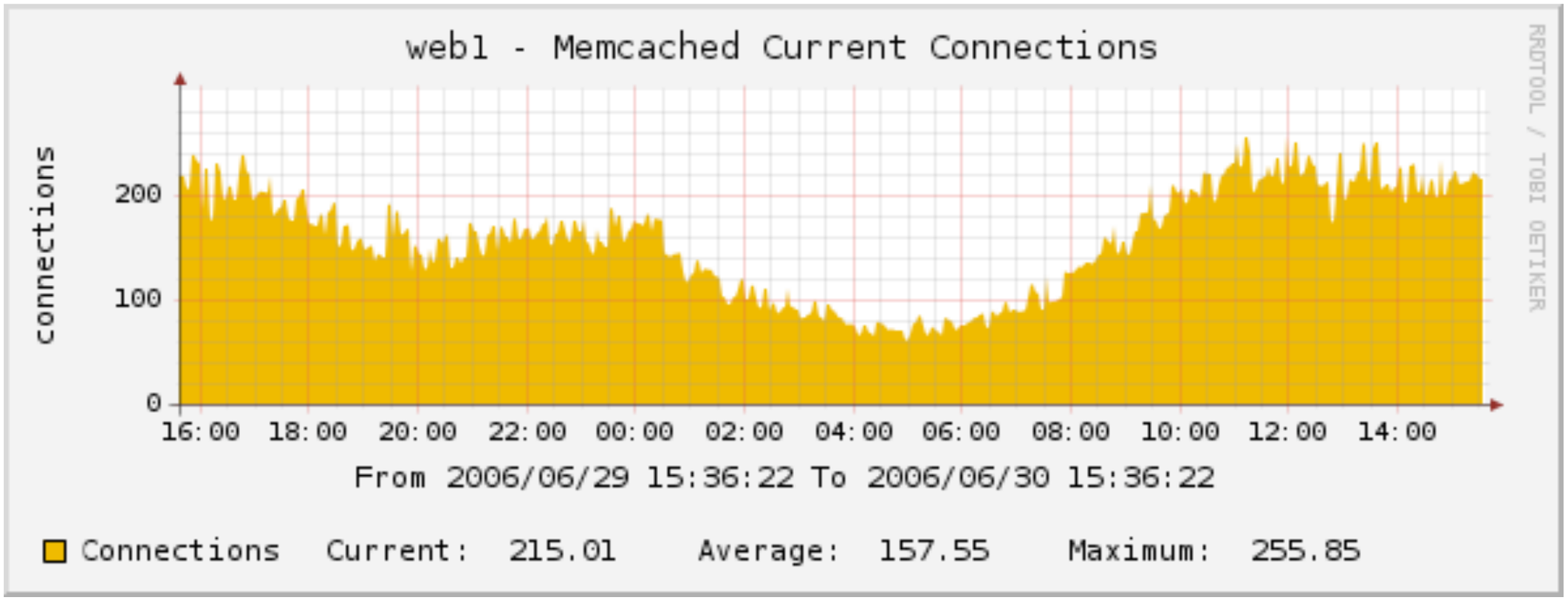

mber of TCP Establishe
20 k
10 k
0
08 10 12 14 16 18 20 22 00 02 04 06 08 10 12 14 16
TCP Established Cur: 10066 Ave: 9990 Max: 10389 Min: 8936
RRDTOOL / TOBI OETIKER


CPU Usage [%]
100
50
0
10 12 14 16 18 20 22 00 02 04 06 08 10 12 14 16
User Cur: 0.6[%] Ave: 0.6[%] Max: 1.5[%] Min: 0.2[%]
Nice Cur: 0.0[%] Ave: 0.0[%] Max: 0.0[%] Min: 0.0[%]
System Cur: 1.2[%] Ave: 1.2[%] Max: 3.0[%] Min: 0.4[%]
Idle Cur: 97.7[%] Ave: 97.5[%] Max: 99.3[%] Min: 94.2[%]
Wait Cur: 0.0[%] Ave: 0.0[%] Max: 0.0[%] Min: 0.0[%]
Kernel Cur: 0.0[%] Ave: 0.0[%] Max: 0.0[%] Min: 0.0[%]
Interrupt Cur: 0.5[%] Ave: 0.6[%] Max: 1.3[%] Min: 0.1[%]
RRDTOOL / TOBI OETIKER

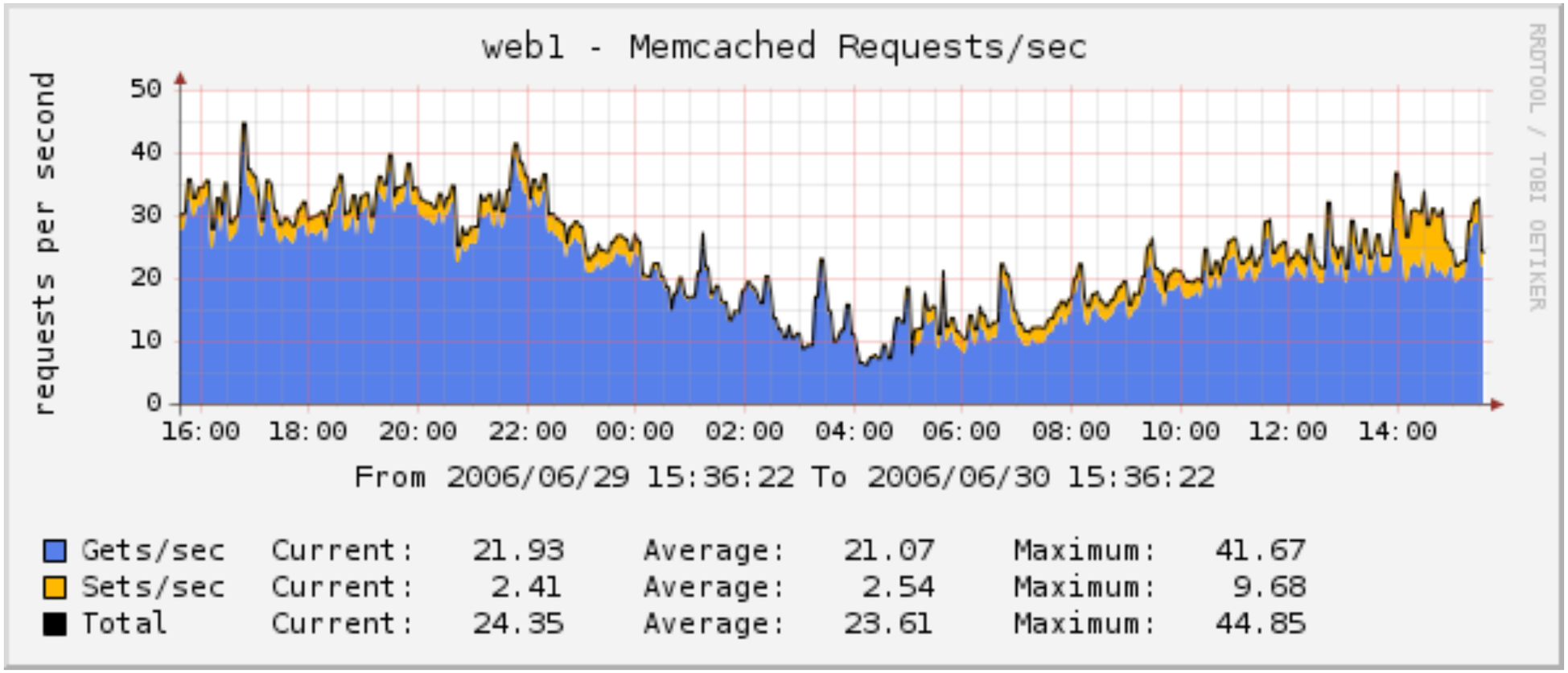
...and others

- Cacti
  - http://dealnews.com/developers/cacti/memcached.html
- Ganglia
  - http://ganglia.wiki.sourceforge.net/

Current Connections



Requests Per Second



1.2.5 Release

- Multi Interface Support
- UDP all the time
- noreply (SET with no response)
- Solaris Support (large memory page support)
- getimeofday() optimizations
- IPV6

Future

- Binary Protocol
- Multiple Engine Support
  - Char based
  - Durable
  - Queue
  - Highly Threaded

More Resources...

- http://danga.com/memcached/
- #memcached on Freenode
- http://tangent.org/552/libmemcached.html
- Memcached Mailing List (currently Danga.com, soon to move)