pgmemcache and the over reliance on RDBMSs

sean@chittenden.org

Tenets of Fast Applications

- *Fast == Good*
- *Slow == Bad*
- *Disk IO == Bad*
- *Memory == Good*
- *Central Point of Failure == Bad*
- *Distributed == Good*
- *Redundant == Good*

RDMS

- *RDBMS incorrectly defined acronym*
- *Relational Data Management Systems*
- *RDBMSs should not be used as RDMSs*
- *RDBMSs generate lots of disk IO*
- *RDBMSs are central points of failure*
- *RDBMSs carry businesses on shoulders*
- *RDBMSs are heavily loaded, most of the time*

Temptation and Sin

- *Temptation to over rely on RDBMSs is great*
- *Great temptation leads to great peril*
- *RDBMSs manage data well*
- *All RDBMSs require disks (ACID)*
- *All RDBMSs are slow*
- *Conclusion: RDBMSs are a sinful indulgence for application developers*

The Good, Bad, and Ugly of RDBMSs

- *RDBMSs are a great place to store data*
- *RDBMSs are a fantastic way to organize data and businesses*
- *RDBMSs are not a fast source for data*
- *RDBMSs bottleneck easily and scale terribly*

80/20: The Hot Potato

- *80% of a database's data is dormant*
- *20% of a database's data is hot, active, and constantly being queried*
- *99% of all applications fetch 100% of their data from databases*
- *99% of applications are abusing a data management system as a data source*
- ***99% of applications don't use a cache as a data source***

Enter Data Caching

- *Busy sites learned long ago: avoid looking up data in the database*
- *Common tricks include distributing load via DNS (ex: user.db.example.com)*
- *Using a local database (ex: cdb, bdb, gdbm, and MySQL)*
- *Stuff data into SysV IPC shared mem*

Problem with Data Caching

- *All applications build their own caching that is normally language specific*
- *Data Expiration*
- *Cache coherency*
- *Only works for reading data, not writing*
- *Invalidating cached data*
- *Displaying wrong data to customers is costly (think Toys'R'Us and crying kids)*

Enter memcached(8)

- *memcached is a flat, distributed data cache*
- *memcached scales infinitely*
- *memcached does not use disk*
- *memcached works well with RDBMSs*
- *memcached is application driven*
- *memcached is **fast** to the tune of ~100+K requests per server per second*
- *memcached **solves** the hot potato problem*

How memcached(8) works

- *Two hash levels*
- *Client hashes key to determine which server stores the key*
- *Server stores key/value in hash*
- *That's it. There ain't no more to it.*

Limitations

- *memcached(8) is not an RDBMS or even an RDMS*
- *memcached(8) is language neutral*
- *memcached(8) has no structure for stored data*
- *memcached(8) only manages the expiration of data and its available space via a LRU algo*
- *memcached(8) has to have its data managed*
- *Text protocol*
- *No spaces in keys*

Limitations: Part Two

- *Server has no knowledge of server lists*
- *Server lists have to be kept in sync across hosts and applications*
- *Server flap kills cache hit rates*
- *Only helps with frequently accessed data*
- *Doesn't help with OLAP applications*
- *Doesn't help with write caching*
- *Relies on a sealed network for security*

libmemcache(3)

mc = **mc_new**(*);*
mc_server_add(*mc, “host1”, “11211”);*
mc_add(*mc, key, key_len, val, val_len,*
expiration, flags);
val = **mc_aget**(*mc, key, key_len);*
mc_replace(*mc, key, key_len, val, val_len,*
expiration, flags);
mc_set(*mc, key, key_len, val, val_len,*
expiration, flags);
mc_delete(*mc, key, key_len, hold);*

libmemcache(3)

req = **mc_req_new**(*);*
key1_res = **mc_req_add**(*req, key1, key1_len);*
key2_res = **mc_req_add**(*req, key2, key2_len);*
mc_get(*mc, req);*
mc_incr(*mc, key, key_len, increment);*
mc_decr(*mc, key, key_len, decrement);*
stats = **mc_stats**(*mc);*

pgmemcache to the rescue

- *PostgreSQL manages data*
- *PostgreSQL* replaces *and* deletes *data in memcached(8)*
- *Applications* add *data to memcached(8)*
- *pgmemcache only lets you use one memcache domain per backend*

Step #1: Write mc_init()

```
CREATE OR REPLACE FUNCTION mc_init()
RETURNS VOID AS 'BEGIN
    IF memcache_init() THEN
        PERFORM memcache_server_add("mc1.example.com", "11211");
        PERFORM memcache_server_add("mc2.example.com", "11211");
    END IF;
    RETURN;
END;' LANGUAGE 'plpgsql';
```

Step #2: Write an Update Function

```
CREATE FUNCTION auth_passwd_upd() RETURNS TRIGGER
AS ‘BEGIN
    IF OLD.passwd != NEW.passwd THEN
        PERFORM mc_init();
        PERFORM memcache_replace(”user_id_” || NEW.user_id || ”_password”, NEW.passwd);
    END IF;
    RETURN NEW;
END;’ LANGUAGE ‘plpgsql’;

CREATE TRIGGER auth_passwd_upd_trg
    AFTER UPDATE ON passwd
    FOR EACH ROW
    EXECUTE PROCEDURE auth_passwd_upd();
```

Step #3: Write a Delete Function

```
CREATE FUNCTION auth_passwd_del() RETURNS TRIGGER
AS 'BEGIN
    PERFORM mc_init();
    PERFORM memcache_delete("user_id_" || OLD.user_id || "_passwd", OLD.passwd);
    RETURN OLD;
END;' LANGUAGE 'plpgsql';

CREATE TRIGGER auth_passwd_del_trg
    AFTER DELETE ON passwd
    FOR EACH ROW
    EXECUTE PROCEDURE auth_passwd_del();
```

pgmemcache API #1

memcache_init*()*

- *Initializes the backend to work with memcached(8). Returns TRUE if this call initialized itself (ie, servers need to be added).*

memcache_server_add*(*/* *server* */* TEXT, /* *port* */* TEXT*)*

- *Adds a server to the list of available servers.*

memcache_add*(*/**key*/* TEXT[, /**value*/* TEXT[, /**expire*/* INTERVAL[, /**flags*/* INT2]]]*)*
memcache_add*(*/**key*/* TEXT, /**value*/* TEXT, /**expire*/* TIMESTAMP WITH TIME ZONE[, /**flags*/* INT2]*)*

- *Adds a key to the cache cluster if the key does not already exist.*

pgmemcache API #2

newval = memcache_decr(/**key**/ TEXT[,
/**decrement**/ INT4])

- *If key exists and is an integer, atomically decrements by the value specified (default decrement is one). Returns value after decrement.*

memcache_delete*(*/**key**/ TEXT*[*,
/**hold timer**/ INTERVAL*])*

- *Deletes a given key. If a hold timer is specified, key with the same name can not be added until the hold timer expires.*

pgmemcache API #3

memcache_flush_all*(/*key*/* TEXT*)*

- *Flushes all keys from the backend as calculated by the passed key.*
-

memcache_free*( )*

- *Cleans up libmemcache from the backend*
-

value = memcache_get*(/*key*/* TEXT*)*

- *Fetches a key out of the cache. Returns a TEXT for keys that are found and NULL for keys that didn't exist. Zero length values are valid.*

pgmemcache API #4

hash = memcache_hash(/**key*/ TEXT)

- *Returns the hash value for a given key*

newval = memcache_incr(/**key*/ TEXT*[*, /**increment*/ INT4*])*

- *If key exists and is an integer, atomically increment by the value specified (default increment is one). Returns value after increment.*

pgmemcache API #5

memcache_replace*(/*key*/* TEXT[, */*value*/* TEXT[,
*/*expire*/* INTERVAL[, */*flags*/* INT2]]]*)*
memcache_replace*(/*key*/* TEXT, */*value*/* TEXT,
*/*expire*/* TIMESTAMP WITH TIME ZONE[, */*flags*/* INT2]*)*

- *Replaces an existing key's value if the key already exists.*

memcache_set*(/*key*/* TEXT[, */*value*/* TEXT[,
*/*expire*/* INTERVAL[, */*flags*/* INT2]]]*)*
memcache_set*(/*key*/* TEXT, */*value*/* TEXT,
*/*expire*/* TIMESTAMP WITH TIME ZONE[, */*flags*/* INT2]*)*

- Regardless of whether the key exists or not, set the value for the key to the specified value.

pgmemcache API #6

stats = memcache_stats*()*

- *Returns a TEXT string with all of the stats from all servers in the server list*

stat = memcache_stats*(/*statistic key*/* TEXT*)*

- *Returns a specific statistic as a TEXT object. Statistic derived from summation of all servers in server list.*

memcached(8) best practices

- *Use servers without hard drives*
- *Use PXE to net boot memcache servers*
- *Use 64 bit architectures (ex: AMD64)*
- *Don't skimp on RAM*
- *If using 32bit architecture, don't use more than 4GB of RAM (don't use PAE: it's evil)*
- *Use FreeBSD* *or Leenox with epoll*

pgmemcache Best Practices

- *Use triggers*
- *Use the LISTEN/NOTIFY mechanism to simulate ON COMMIT triggers*
- *Use manageable keys*
- *Help fund someone to add ON COMMIT triggers to PostgreSQL *grin**

Application Development Advice

- *Convert small portions of an application at a time using XP practices*
- *Many small keys work well*
- *Avoid caching serialized data structures (eg: PHP, Java, C struct's, etc.)*
- *RAM is a bigger constraint than CPU in nearly all memcached(8) installations*
- *It's easy to update small single key/values pairs (user => password, username => host/port)*
- *It's hard to update complex key/value pairs*

Suggestions

- *Perdition*
- *Postfix*
- *BIND+DLZ*
- *Apache/mod_* (duh!)*
- *Authentication on trusted networks*

Q&A

Ask ‘em if you’ve got ‘em!

“If you ask a stupid question, you may feel stupid. If you don’t ask a stupid question, you remain stupid.”

-Tony Rothman, Ph.D.U. Chicago, Physics

Thanks!

http://people.FreeBSD.org/~seanc/libmemcache/
http://people.FreeBSD.org/~seanc/pgmemcache/
e: sean@chittenden.org

Consulting to integrate libmemcache or pgmemcache into applications, commercial support, and training are available. Please send email for details.