Benchmarking of Web tools

1 Configuration

Tests have been run on two servers.

Server 1:

- nginx 1.0.6, CT++ module (http://ngx-ctpp.vbart.ru/),
- php-fpm 5.3 + APC (http://php.net/manual/en/book.apc.php)
- Yii 1.1.8 (http://www.yiiframework.com/)
- Node.js 0.4.11 (http://nodejs.org/)

Server 2:

- nginx 1.0.6, CT++ module (http://ngx-ctpp.vbart.ru/),
- php-fpm 5.3 + APC
- Node.js 0.4.11 (http://nodejs.org/)
- MongoDB 2.0 (http://www.mongodb.org/downloads)
- Redis 2.2.12 (http://redis.io/)
- PostgreSQL 9.1 (http://www.postgresql.org/)

1. For CT++, please specify:
   - templates_root – a directory where it will load the template (locally)
   - ctpp2_data_buffer 512K
4. In nginx, enable php-fpm for the test site, e.g., /var/www/yii/site
5. Allocate at least 200Mb for APC cache, apc.stat = 0
5. Put Yii one level higher than the test site doc_root. For instance, /var/www/yii/framework and /var/www/yii/site
6. Similarly to Yii, allocate a directory for node.js scripts. For instance, /var/www/yii/framework and /var/www/yii/site
7. Using npm under node.js, install the following modules:
   - Socket.io (https://github.com/learnboost/socket.io)
   - Redis + hiredis
   - MongoDB
8. Create a database in PostgreSQL. For testing purposes, you can use the Root access

2 Test results

Tools:

1. apache benchmark (ab):
   run with -n 5000 -c 100
2. siege:
   run with-d0 - r5000 -c100

3. httperf:
   run with --hog --wsess 1000,1000,0.01 --rate 50 --timeout 5 --method GET and --hog --wsess 1000,1000,0.1 --rate 50 --timeout 5 --method GET

2.1 Comparative analysis of the basic frameworks initialization time

- Yii (1.1.8): time to process a minimum request: 27 to 47ms, avg: 34ms
- Symfony (2.0.1): time to process a minimum request: 46 to 92ms, avg: 72ms
- ZF (optimized build): time to process a minimum request: 114 to 201ms, avg: 132ms

Conclusion: Yii has a minimal initialization time and shows better than acceptable response time with requests from cache or small requests.

2.2 Comparative analysis of the basic template engines

Template engine	CT + + (2.7.1)	Smarty (3.10)	Twig (1.1.2)
A simple output of 3 variables			
single call without a framework	4-5ms	6-7ms	6-7ms
- at the first access	4-5ms	14ms	12ms
Output of table of 1000 rows and 3 columns			
at the first access	33ms	112ms	110ms
subsequent accesses	33ms	79ms	82ms
Lebowski benchmark (building of a typical page with a header, left navigation column, summaries, links to full version, tag cloud and footer)			
single request at the first access	16ms	77ms	82ms
single request at subsequent accesses		27ms	28ms
under load	27ms	79-101ms	82-104ms
requests per second	1531	1213	1191
memory usage at full load	324Mb	387Mb	372Mb

Impressions from the template engines:

- CT++ - the templates are completely separated from the application. All the templates are pre-compiled, and you can easily include the compilation procedure in a deploy script. The language of templates offers a feature set similar to Smarty. Still, it is somewhat more cumbersome because of structures like <TMPL_var total>, <TMPL_if sections>: in Smarty this corresponds to {$ total}, {if $ sections}
- Smarty 3 is more familiar to users and it has an extended syntax now. It is easy to use, and it can be integrated into the

framework
- Twig is similar to a Smarty 3, but it is more concise and user-friendly in terms of design, and it can be integrated into the framework

Conclusion: CT++ is the quickest template engine, it uses fewer resources and better withstands the load. It allows you to completely separate presentation logic from your application. Also, you will be able to unify output of your application. Also, an important benefit of CT++ is that it is agnostic of programming language: you can make output templates, for instance, for php applications and node.js. Twig and Smarty are more familiar to users and perhaps also more convenient.

2.3 Verifying a typical response scenario

Scenario:

1. Receive HTTP GET request from a client
2. Send it to Yii for execution
3. Select the required blocks from the cache
4. Provide CT++
5. Return the result to the client

Tools:

apache benchmark (ab): run with -n 5000 -c 100

- data selection in one query to the cache: full time of the request - 14ms, under load - 66ms
- blockwise data selection and merger into a final json (5 blocks): 14ms, under load - 72ms, 1222 requests/s
- blockwise data selection and merger into a final json (45 blocks): 14ms, under load - 79ms, 1151 requests/s

Conclusion: The testing results show that for a typical page structure, page build time does not exceed 80ms, if all the data needed is in cache. Such a system is potentially capable of processing 1200 requests per second or 4.3 million per hour, so you can avoid HTML caching and focus on data caching.

2.4 Testing of fault-tolerance of mongodb replica set

Scenario:

For the test, 3 mongodb instances were launched:

- /data/service/mongodb/bin/mongod --rest --replSet ntv --port 27017 --dbpath /data/r0

- /data/service/mongodb/bin/mongod --replSet ntv --port 27018 --dbpath /data/r1
- /data/service/mongodb/bin/mongod --replSet ntv --port 27019 --dbpath /data/r2

After configuring a replica, ports 27017 and 27019 were sequentially disconnected. After each mongodb instance disconnection, we checked the status of remaining servers and their availability to PHP. The status of the master instance was promptly changing, and the data retrieved from the database was up-to-date. After the master database failure, the client is automatically connected to a new master in the replica set. Here is a full connect string:

```
$ M = new Mongo ("mongodb: / /
193.232.148.35:27017,193.232.148.35:27018,193.232.148.35:27
019", array ("replicaSet" => true));
```

With such a connect string, server failure on ports 27017 or 27019 has no effect on the client operation.

The following connect string is also possible:

(master - 193.232.148.35:27018):

```
$m = new Mongo("mongodb://193.232.148.35:27017",
array("replicaSet" => true));
```

In such a case, the system automatically connects to the master instance (detected automatically), but in case 193.232.148.35:27017 is disconnected, the "Transport endpoint is not connected" error is thrown on reconnect.

Here are the cons of the solution:

1. Oplog puts a limitation on the collection size, so if the master instance has been down for a long time, it will not be able to synchronize on its own as older oplog collection records could be replaced by the newer records. This is not actually an issue, as there is a synchronization command, which does not rely on oplog and enables load balancing. So, just run {resync:1} command on the slave or run the slave with the --autoresync key.
2. At manual shutdown of one of the instances, the lag of php drivers is observed and the "couldn't send command" error occurs. However, after reloading the page, it disappears. In this

case, if you shut down the instance and restart php-fpm, the error does not occur at all. Apparently, the error occurs as a consequence of caching of the master server PHP address. Connect to the replica via a Mongo console is completely transparent, so you would not notice instance failure. Perhaps this behavior is a consequence of mongodb instance separation by ports rather than hosts. Still, in the current configuration, verifying of this is not realistic.

2.5 Testing mongodb

1. Insert of 10,000,000 records of Array ([_id] => MongoId Object ([$id] => 4e8d2fe42b6ac6d41b0000a3) [i] => 163 [status] => 1 [x] => 8973 [y] => 20433) individually - 392 sec
2. Insert of 10,000,000 records of Array ([_id] => MongoId Object ([$id] => 4e8d2fe42b6ac6d41b0000a3) [i] => 163 [status] => 1 [x] => 8973 [y] => 20433) in blocks of 10,000 records - 392 sec
3. Update of all records in the collection with an increment of "y" to 1 - 927 sec
4. At the time of the update, 10 records are inserted independently to the same collection. As a result of the test, the data was saved successfully, and the counters were updated
5. Selection at the time of the update: out of 3 attempts 2 attempts were successful, and one has failed due to a timeout.