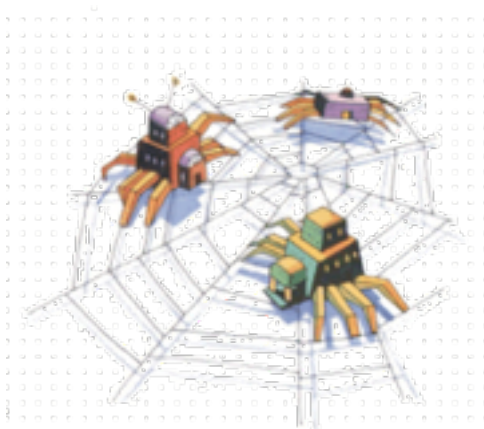
Crawler Distribution using redis

Benjamin Schiller, P2P Networks, TU Darmstadt

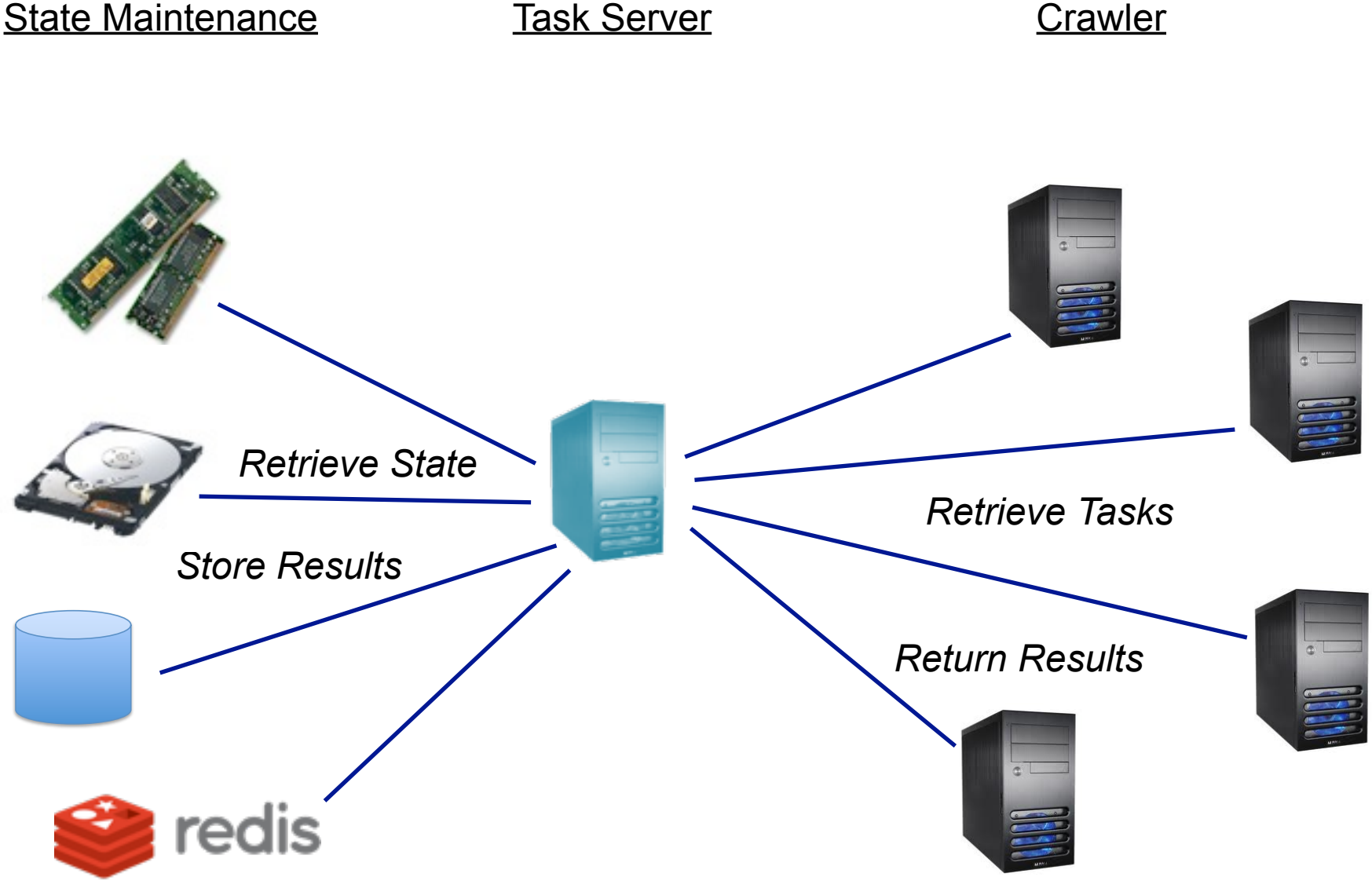
Network Tech Talk Series / A213 / 04.01.2011



Crawler Bottlenecks

- CPU
  - Post-process [if possible]
- Memory
  - Post-process [if possible]
  - Write to HD [if possible]
- Network
  - Bandwidth not a big problem
  - BUT
    - Delays
    - Number of concurrent connections
    - Being blocked by the service provider

Crawler Distribution



Redis

- An in-memory ‘key-value store’ / ‘data structure server’
- Values can be:
  - Strings, lists, sets, sorted sets, hashes
- Atomic operations to:
  - Insert key/value pairs, append to lists, insert into sets, etc.
- Bindings available for many languages:
  - ActionScript, C, C++, C#, Clojure, Common Lisp, Erlang, Go, Haskell, haXe, Io, Java, server-side JavaScript (Node.js), Lua, Objective-C, Perl, PHP, Pure Data, Python, Ruby, Scala, Smalltalk, Tcl.
- Developed by Citrusbyte, sponsored by VMware
  - Licenced under BSD
  - http://redis.io/
  - Current version: 2.4.5

Redis - Strings

- Arbitrary data, max 512 MB
- Atomic counters, bit arrays
- Commands
  - DECR, DECRBY, INCR, INCRBY
  - GET, GETBIT, GETRANGE, GETSET
  - SET, SETEX, SETNX, SETBIT, SETRANGE, STRLEN, APPEND
  - MGET, MSET, MSETNX

```
redis > SET bla "abc"
OK
redis > APPEND bla "defg"
(integer) 7
redis > GETSET bla "xyz"
"abcdefg"
```

```
redis > INCR fasel
(integer) 1
redis > INCRBY fasel 10
(integer) 11
redis > DECR fasel
(integer) 10
```

Redis - Lists

- List of strings, ordered by insertion order (L = head, R = tail)
- Max length of a list: 2^{32}-1 = 4,294,967,295
- Commands:
  - LPOP, LPUSH, LPUSHX, RPOP, RPUSH, RPUSHX, RPOPLPUSH
  - BLPOP, BRPOP, BRPOPLPUSH
  - LINSERT, LSET, LLEN, LRANGE, LREM, LTRIM, LINDEX

```
redis > RPUSH myList first
(integer) 1
redis > RPUSH myList second
(integer) 2
redis > LPUSH myList third
(integer) 3
redis > LRANGE myList 0 -1
1) "third"
2) "first"
3) "second"
```

```
redis > RPOPLPUSH myList myList2
"second"
redis > LRANGE myList 0 -1
1) "third"
2) "first"
redis > LRANGE myList2 0 -1
1) "second"
```

Redis - Sets, Sorted Sets, Hashes

- Sets
  - Unordered, non-repeated collection of Strings (max size 2^{32}-1)
  - Add, remove, test for existence: O(1) (!!!)
  - *SADD, SINTER, SUNION, SDIFF, SMEMBERS, SPOP, ...*
- Sorted Sets
  - Ordered, non-repeated collection of Strings
  - Add, remove, test for existence: O(log N)
  - *ZADD, ZREM, ZSCORE, ZRANK, ZCOUNT, ...*
- Hashes
  - Maps between String fields and String values
  - *HMSET user:1000 username antirez password P1pp0 age 34*
  - *HGET, HDEL, HLEN, HSET, ...*

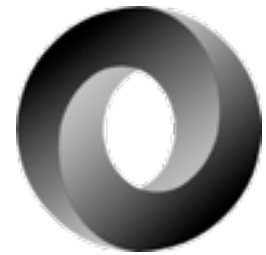
Other Commands

- **AUTH** password
- **SUBSCRIBE** channel
- **PUBLISH** channel message
- **FLUSHALL** / **FLUSHDB**
- **TYPE** key
- ...

- http://redis.io/commands

Tasks / TaskLists


```
"{
   started: -1,
   series: 12-01-02-12-41,
   index: 169,
   crawler: null,
   tasks: [
      {
         count: -1,
         url: http: //www.heise.de/newsticker/meldung/Piratenpartei-Chef-Traumkoalition-mit-FPD-und-Gruenen-1402256.html,
         page_id: 2,
         content: null,
         info_type: GP,
         retrieved: -1,
         article_id: 121
      },
      {
         count: -1,
         url: http: //www.heise.de/newsticker/meldung/Jonathan-Zittrain-bedauert-den-Tod-des-PC-1401319.html,
         page_id: 2,
         content: null,
         info_type: FB,
         retrieved: -1,
         article_id: 302
      }
   ]
}"
```

Data Structures

- npc:tasks:tasklist:index (String)

- npc:tasks:tasklist:${index} (String)
- npc:tasks:tasklist:crawler:${index} (String)
- npc:tasks:tasklist:started:${index} (String)

- npc:tasks:queue (List)
- npc:tasks:ongoing (List)
- npc:tasks:completed (List)

Task Distribution - Workflow

1. Add new taskList

2. Retrieve taskList

3. Process taskList

4. Process completed taskLists

5. Re-queue ongoing taskLists

Task Distribution - Workflow (1)

1. Add new taskList
   - create task_list (JSON)
   - index = **INCR** npc:tasks:tasklist:index
   - **SET** npc:tasks:tasklist:${index} ${task_list}
   - **LPUSH** npc:tasks:queue ${index}
   - .

Task Distribution - Workflow (2)

2. Retrieve taskList
   - index = **RPOPLPUSH** npc:tasks:queue npc:tasks:ongoing
   - task_list = **GET** npc:tasks:tasklist:${index}
   - task_list.started = ${current_timestamp}
   - task_list.crawler = ${name}
   - **PUT** npc:tasks:tasklist:started:${index} ${current_timestamp}
   - **PUT** npc:tasks:tasklist:crawler:${index} ${name}
   - .

Task Distribution - Workflow (3)

3. Process taskList

- task.retrieved = ${current_timestamp}
- task.count, task.content = ${count}, ${content}
- started_ = **GET** npc:tasks:tasklist:started:${index}
- IF (${started} is None OR ${started_} != ${started}) ; break
- **SET** npc:tasks:tasklist:${index} ${task_list}
- **LPUSH** npc:tasks:completed ${index}
- **LRM** npc:tasks:ongoing 0 ${index}
- **DEL** npc:tasks:tasklist:started:${index}
- **DEL** npc:tasks:tasklist:crawler:${index}
- .

Task Distribution - Workflow (4)

4. Process completed taskLists
   - def getCompletedTaskList()
     - index = **LPOP** npc:tasks:completed
     - if index is None : return None
     - task_list = **GET** npc:tasks:tasklist:${index}
     - **DEL** npc:tasks:tasklist:${index}
     - return task_list
   - task_list = getCompletedTaskList()
   - while not taskList is None :
     - for task in task_list.tasks :
       - db.insertInfo(task)
     - task_list = getCompletedTaskList()
     - .

Task Distribution - Workflow (5)

5. Re-queue ongoing taskLists
   - indexes = **LRANGE** npc:tasks:ongoing 0 -1
   - for index in indexes :
     - started = **GET** npc:tasks:tasklist:started:${index}
     - crawler = **GET** npc:tasks:tasklist:crawler:${index}
     - if (started == none OR crawler == none) ; continue
     - **DEL** npc:tasks:tasklist:started:${index}
     - **DEL** npc:tasks:tasklist:crawler:${index}
     - **LREM** npc:tasks:ongoing 0 ${index}
     - **RPUSH** npc:tasks:queue ${index}
     - .

Summary

- Crawler / task distribution done easily
- No need for a separate task distribution server
- Less overhead than SQL-based solutions
- Very fast, 30k - 150k requests per second
- Drawback: no persistence of storage [without overhead]