What the root cause of service outage?

[SoCC Why Does the Cloud Stop Computing?]

Dataset

- Google.com/bing.com
- Query: “serviceName outage month year”
- January2009 to December2015
- 1247 unique links describing 597 outages

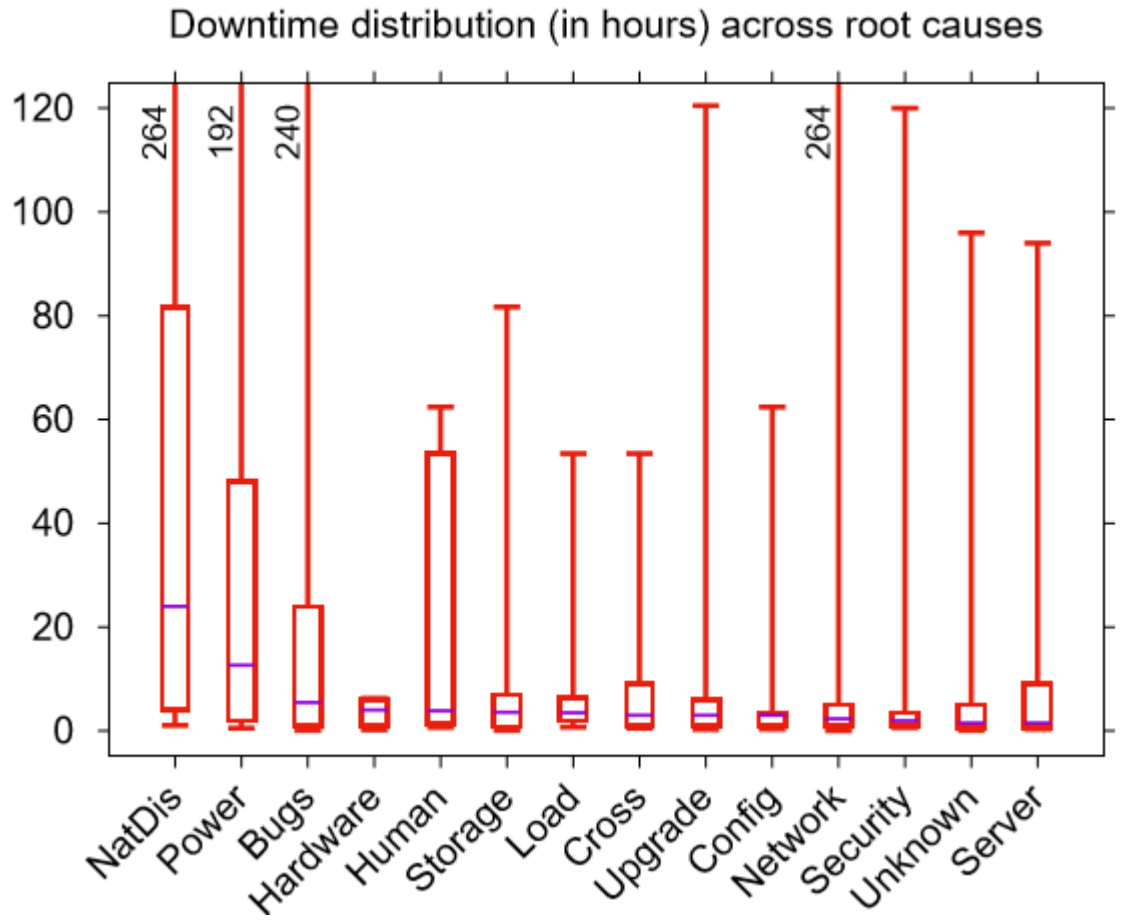
Category	Service Names
CH: Chat	Blackberry Messenger, Google Hangouts, Skype, WeChat, WhatsApp
EC: E-Comm.	Amazon.com, Ebay
ML: Email	GMail, Hotmail, Yahoo Mail
GM: Game	PS Network, Xbox Live
PA: PaaS/IaaS	Amazon EBS, EC2, and RDS, Google Ap-pengine, Microsoft Azure, Rackspace
SA: SaaS	Google Docs, Office365, Salesforce
SC: Social	Facebook, Google Plus, Instagram, Twitter
DT: Storage	Apple iCloud, Box, Dropbox, Google Drive, Microsoft SkyDrive
VD: Video	Netflix, Youtube

What the root cause of service outage?

[SoCC Why Does the Cloud Stop Computing?]

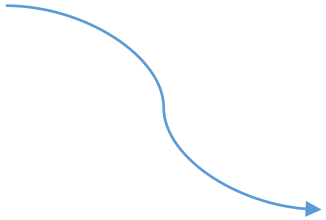
Root cause statistics

§	Root cause	#Sv	Cnt	%	Cnt '09-'15
	UNKNOWN	29	355	-	M.M.M.M.M.M.M
5.1	UPGRADE	18	54	16	7.4.M.5.M.4.7
5.2	NETWORK	21	52	15	4.4.6.8.M.8.5
5.3	BUGS	18	51	15	M.4.9.8.9.9.2
5.4	CONFIG	19	34	10	2.2.7.2.5.M.4
5.5	LOAD	18	31	9	2.5.5.5.4.8.2
5.6	CROSS	14	28	8	-.2.4.M.5.3.4
5.7	POWER	11	21	6	5.4.3.5.3.1.-
5.8	SECURITY	9	17	5	7.-.2.1.3.4.-
5.9	HUMAN	11	14	4	-.1.4.4.2.1.2
5.10	STORAGE	4	13	4	2.-.-.3.5.3.-
5.11	SERVER	6	11	3	-.3.-.2.2.4.-
5.12	NATDIS	5	9	3	1.1.3.2.1.1.-
5.11	HARDWARE	4	5	1	1.-.-.3.1.-.-



Root Cause

- Upgrade
- Network
  - ✓ Broken hardware(switches die simultaneously)
  - ✓ Networking layer(access misconfiguration)
  - ✓ SDN control cluster
  - ✓ Sofware bugs(Traffic control, routing loops, memory allocation)
- Bugs
- Configuration



Latent configuration errors

- ✓ Such configuration parameters are neither used nor checked during normal operations, errors in their settings go undetected until their late manifestation
- ✓ Often associated with fail-over, error handling, backup, load balancing, mirroring.

Google App Engine Outage Timeline(2010)

7:48 AM – Internal monitoring graphs first began to show elevated errors in the primary data center.

7:53 AM – The on-call staff was notified that the primary data center had suffered a power outage and that about 25% of the servers had not received backup power.

8:01 AM – The primary on-call engineer determined that the Google App Engine was down. He paged product managers and engineering leads, requesting them to handle communication with the users about the outage.

8:22 AM – Though power had been restored to the data center, it was determined that many servers were down and that the surviving servers were not able to handle the traffic. Major clusters had lost enough machines that they were not able to carry the load. The on-call team agreed to invoke the unexpected failover procedure for an unplanned data center outage.

8:40 AM – Two conflicting sets of procedures were discovered. The team attempted to contact the specific engineers responsible for procedure changes so that the situation could be resolved.

8:44 AM – The primary on-call engineer attempted to move all traffic in a read-only state to the backup data center. Unexpected configuration problems from this procedure prevented the read-only backup from working properly.

9:08 AM – New data seemed to indicate that the primary data center had recovered. With no clear policy, the team was not aware that based on historical data, the primary data center was unlikely to have recovered to a usable state. An attempt was made to move traffic back to the primary data center while the read-only problems in the backup data center were debugged.

9:18 AM – The primary on-call engineer determined that the primary data center had not recovered. Traffic was failed back to the backup data center, and the unplanned failover procedure was reinitiated.

9:35 AM – An engineer familiar with the unplanned failover procedure was finally reached and began providing guidance about the procedure. Traffic was moved in read-only mode to the backup data center.

9:48 AM – Read-only mode became operational. Applications that handled read-only mode worked properly but in a reduced operational mode.

9:53 AM – Relevant engineers were now online. The correct procedure document was confirmed. The actual failover procedure for reads and writes began.

10:08 AM – The unplanned failover procedure completed with no problems. The App Engine was restored to service.

Real world Case

LC error from mapreduce

1. Configuration error:

`mapred.local.dir`
= directory path w/ wrong owner

(`mapred.local.dir` is not used until exec. of MapReduce jobs)

2. Impact

The TaskTrackers were trapped into infinite loops ("When I ran jobs on a big cluster, some map tasks never got started.")

3. Code snippets: `/* TaskTracker.java */`

```
// no check at initialization
while (running) {
  try {
    ...
    access mapred.local.dir
    ...
  } catch(Exception e) {
    LOG.log("Retrying!");
  }
}
```

Infinite loops

Throw Exception

Too late to avoid the failure!

User requests: *"TaskTracker should check whether it can access to the local dir at the initialization time, before taking any tasks."*

LC error from Squid

Configuration error:
`diskd_program` = a non-existent path

Parse config files; store the settings in program vars.

Use the setting of `diskd_program` for log rotation.

Diagnosis (48 hrs)
- **26** rounds of diagnostic conversations;
- **5** collections of logs & runtime traces;
- **2** incorrect patches.

[Patch] Check existence of diskd_program during initialization

Software	Description	Lang.	# Parameters	
			Total	RAS
HDFS	Dist. filesystem	Java	164	44
YARN	Data processing	Java	116	35
HBase	Distributed DB	Java	125	25
Apache	Web server	C	97	14
Squid	Proxy server	C/C++	216	21
MySQL	DB server	C++	462	43

Software	Deficiency of initial checking		Studied param.
	Missing	Incomplete	
HDFS	41 (93.2%)	3 (6.9%)	44
YARN	29 (82.9%)	5 (14.3%)	35
HBase	18 (72.0%)	5 (2.0%)	25
Apache	4 (28.6%)	2 (14.3%)	14
Squid	9 (42.9%)	4 (19.0%)	21
MySQL	6 (14.0%)	6 (14.0%)	43

Table 4: Number of configuration parameters that do not have any initial checking code ("missing") and that only have partial checking and thus cannot detect all potential errors ("incomplete").

Software	# RAS Parameters		
	Subject to LC errors		Studied
HDFS	17	(38.6%)	44
YARN	9	(25.7%)	35
HBase	3	(12.0%)	25
Apache	3	(21.4%)	14
Squid	3	(14.3%)	21
MySQL	2	(4.7%)	43
Total	**37**	**(20.3%)**	**182**

Findings

- Many (14.0%–93.2%) of the studied RAS parameters do not have any special code for checking the correctness of their settings. Instead, the correctness is verified (implicitly) when the parameters' values are actually used in operations such as a file open call.
- Many (12.0%–38.6%) of the studied RAS configuration parameters are not used at all during the system's initialization phase.
- Resulting from Findings 1 and 2, 4.7%– 38.6% of the studied RAS parameters do not have any early checks and are thereby subject to LC errors which can cause severe impact on the system's dependability.

Auto-failover configuration parameters: HDFS-2.6.0

`dfs.ha.fencing.ssh.connect-timeout`
`dfs.ha.fencing.ssh.private-key-files`

1. LC Errors:

Ill-formatted numbers (e.g., typos) for ssh timeout;
Invalid paths for private-key files (e.g., non-existence, permission errors).

2. Initial checks: **None**.

3. Late execution: Parse the timeout setting to an **integer** value;
Read the file specified by the key-files setting.

```
public boolean tryFence(...) {
  ...
  int timeout = getInt("dfs.ha.fencing.ssh.connect-timeout");
  ...
  session.createSession();
  ...
}

/* hadoop-common/.../ha/
SshFenceByTcpPort.java */
```

call

```
getString("dfs.ha.fencing.ssh
.private-key-files")

fis = new FileInputStream(prvFile);
```

4. Manifestation:

IllegalArgumentException (when parsing timeout to an integer)
IOException (when reading the key file)

5. Consequence:

HDFS auto-failover fails, and the entire HDFS service becomes unavailable.

(a) Missing initial checking

Error-handling configuration parameter: Apache httpd-2.4.10

`CoreDumpDirectory`

1. LC Errors:

The running program has no permission to access coredump directory.

2. Initial checks: Check if the path points to an **existent directory**.

```
if (apr_stat(&finfo, fname, APR_FINFO_TYPE) != APR_SUCCESS)
  return "CoreDumpDirectory does not exist";
if (finfo.filetype != APR_DIR)
  return "CoreDumpDirectory is not a directory";
```

3. Late execution: Change working directory (**chdir**) to the path.

```
static void sig_coredump(int sig) {
  ...
  apr_filepath_set(ap_coredump_dir, ...);
  ...
}
/* server/mpm_unix.c */
```

"CoreDumpDirectory"

call

```
if(chdir(rootpath) != 0)
  return errno;
```

4. Manifestation:

Error code returned by the `chdir` call

5. Consequence:

Apache httpd cannot switch to the configured directory, and thus fails to generate the coredump file upon server crashing.

(b) Incomplete initial checking

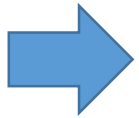
IDEA: Generate configuration checking code from existing source code that use configuration value

- Generate checkers(configuration-consuming instructions & context) for every configuration parameter
- Checkers emulate how the system use configuration value in the original execution

- Invoke checkers at initialization phase of the system(via insertion into bitcode/bytecode)

- Capture anomalies during emulated execution as the evidence of configuration errors

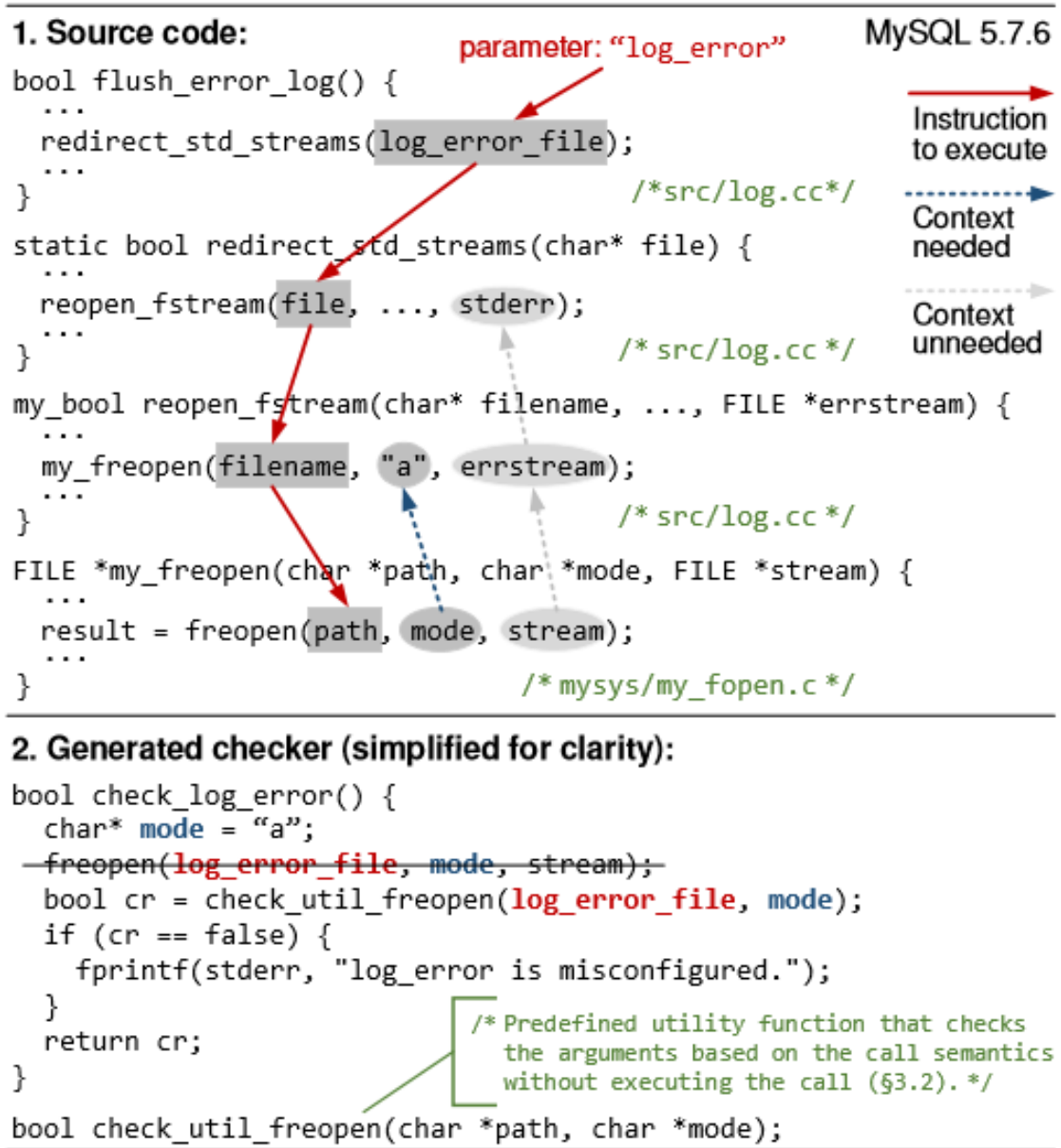
- PCHECK
  - ✓ Based on LLVM and Soot framework
  - ✓ Works on the intermediate representation(IR)of the Program(LLVM IR or Soot Jimple)
  - ✓ Input:
    - ☐ specifications of the configuration interface(API, parser functions, data structures)
    - ☐ Annotations of system initialization phase



Three challenges
1.How to automatically emulate the execution that uses configuration values?

2.Will it be safe?

3.What anomalies should be capture?



Emulate the execution

- Starting point: automatically identify program variables that load configuration parameter based on specified interface
- Using taint tracking method to extracts the instructions that propagate, transform and use the configuration parmeter
- Determine "dependent variables" to produce self-contained context
  - ✓ Backtracks each undefined dependent variable intra-procedurally and inter-procedurally
  - ✓ Doesn't include dependent variables whose values come from indeterminate global varibles, external inputs(I/O, network..)

- Attempts to produce the minimal context necessary to emulate the execution(aware of instruction rewritten)

1. Source code: MySQL 5.7.6

parameter: "log_error"

```
bool flush_error_log() {
  ...
  redirect_std_streams(log_error_file);
  ...
}                                          /*src/log.cc*/

static bool redirect_std_streams(char* file) {
  ...
  reopen_fstream(file, ..., stderr);
  ...
}                                          /* src/log.cc */

my_bool reopen_fstream(char* filename, ..., FILE *errstream) {
  ...
  my_freopen(filename, "a", errstream);
  ...
}                                          /* src/log.cc */

FILE *my_freopen(char *path, char *mode, FILE *stream) {
  ...
  result = freopen(path, mode, stream);
  ...
}                                          /* mysys/my_fopen.c */
```

Instruction to execute

Context needed

Context unneeded

2. Generated checker (simplified for clarity):

```
bool check_log_error() {
  char* mode = "a";
  freopen(log_error_file, mode, stream);
  bool cr = check_util_freopen(log_error_file, mode);
  if (cr == false) {
    fprintf(stderr, "log_error is misconfigured.");
  }
  return cr;
}

/* Predefined utility function that checks
   the arguments based on the call semantics
   without executing the call (§3.2). */
bool check_util_freopen(char *path, char *mode);
```

Will it be safe? Preventing side effects:

- Can't blindly execute original instructions, because it may change internal program state or external system environment(exec, creat or delete files..)
- Internal side effects ar prevented by design
  - ✓ Copy global variables to local ones
  - ✓ Doesn't manipulate pointers if the pointed values are indeterminate.
- External side effects are prevented by rewriting the original call instruction to redirect the calls to predefined check utilities

- Check utilites models a specific system or library call based on the call semantics. It validate the arguments of the call, but doesn't actually execute the call(file access and stats, IP address reachability..)
- PCHECK implements check utilities for syscall, libc functions for C, java core package defined in SDK

1. Source code: MySQL 5.7.6

parameter: "log_error"

```
bool flush_error_log() {
  ...
  redirect_std_streams(log_error_file);
  ...
}                                          /*src/log.cc*/

static bool redirect_std_streams(char* file) {
  ...
  reopen_fstream(file, ..., stderr);
  ...
}                                          /* src/log.cc */

my_bool reopen_fstream(char* filename, ..., FILE *errstream) {
  ...
  my_freopen(filename, "a", errstream);
  ...
}                                          /* src/log.cc */

FILE *my_freopen(char *path, char *mode, FILE *stream) {
  ...
  result = freopen(path, mode, stream);
  ...
}                                          /* mysys/my_fopen.c */
```

Instruction to execute

Context needed

Context unneeded

2. Generated checker (simplified for clarity):

```
bool check_log_error() {
  char* mode = "a";
  freopen(log_error_file, mode, stream);
  bool cr = check_util_freopen(log_error_file, mode);
  if (cr == false) {
    fprintf(stderr, "log_error is misconfigured.");
  }
  return cr;
}

/* Predefined utility function that checks
   the arguments based on the call semantics
   without executing the call (§3.2). */
bool check_util_freopen(char *path, char *mode);
```

What anomalies should be captured?

- Runtime exceptions that disrupt the emulation execution
- Error code returned by system and library calls
- Abnormal program termination, error logging

Invoking Checkers

- for server system, the checkers should be invoked before the server starts to listen and wait for client request
- For distributed system, the checker should be invoked before system starts to connect and join the cluster

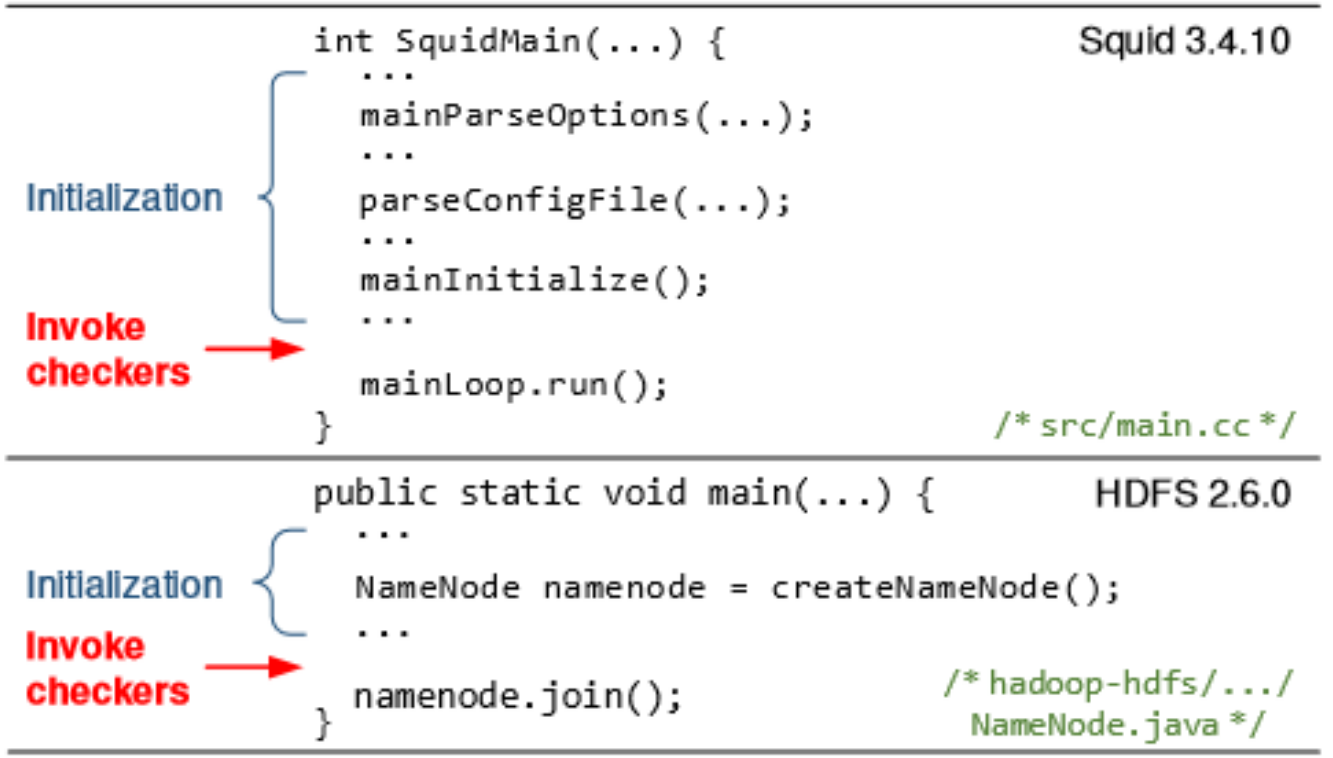


Figure 5: Locations to invoke the checkers in Squid and HDFS NameNode. The auto-generated checkers are expected to be invoked at the end of the initialization phase.

Target at basic & common latent errors.