Understanding, Detecting and Localizing Partial Failures in Large System Software

论　　文　　总　　结　　汇　　报

汇报人：李解

目录

01.

动机和主要工作

Motivation and Main Work

1.1 动机

1. 局部故障经常发生在云系统并造成严重的损失，但是人们对局部故障的认识不足而且也没有有效的检测手段。例如Cassandra开发者采取了非常先进的 accrual故障检测器，但是在处理局部故障时仍然作用很小

2. 局部故障是许多灾难性停机的原因，例如微软365邮件服务就因为服务器的抗病毒引擎模块在识别可疑邮件时卡住导致服务器停机八小时

3. 一旦局部故障发生，由于日志信息中几乎没有哪里出错的信息，所以开发人员需要花费大量时间去找到bug

1.2 主要工作

1. 首先，从五个开源软件（ZooKeeper，Cassandra，HDFS，Apache，Mesos ）中找出100个局部故障案例进行研究，并得出八个发现

2.开发了一个名为OmegaGen的工具，通过静态分析给定系统软件的源码，使用程序简化技术生成该程序的简化版本，这个简化版本称作watchdog，利用watchdog暴露原程序的潜在故障

3. 通过六个大型分布式系统（ZooKeeper，Cassandra，HDFS，HBase，MapReduce，Yarn）评估OmegaGen，在这些系统22个局部故障案例中，生成的watchdog能够检测出20个并且精确找到18个案例的位置。

02.

理解局部故障

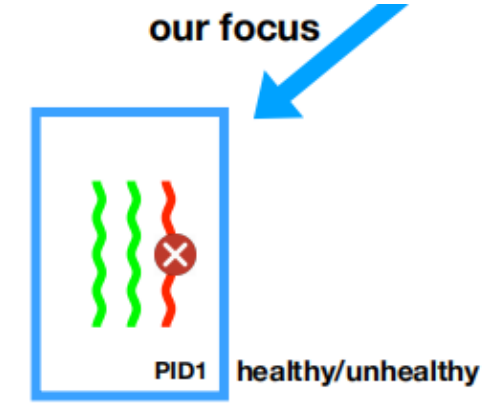
Understanding Partial Failures

2.1 研究范围

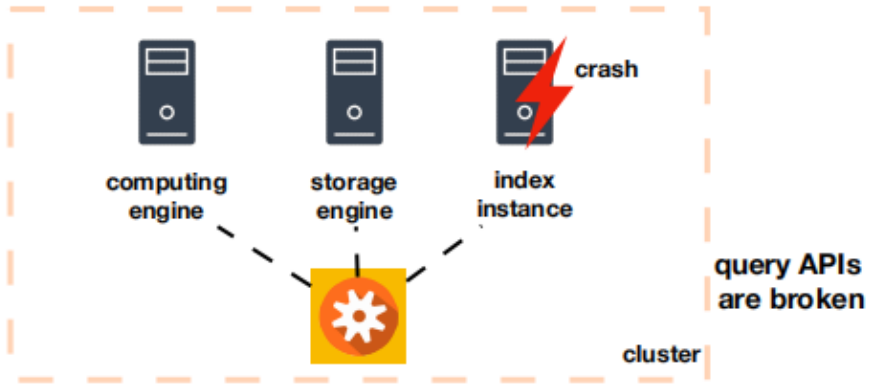
什么是局部故障（Partial Failures）？

- 一个进程中的故障，该故障没有使进程崩溃（crash）但会破坏安全性和活性，或者造成部分功能（非全部功能）的严重迟缓

So：研究局部故障要以进程为粒度而不是服务为粒度，因为服务层的某些故障是由crash造成的。



Process-level



Service-level

2.2 研究发现

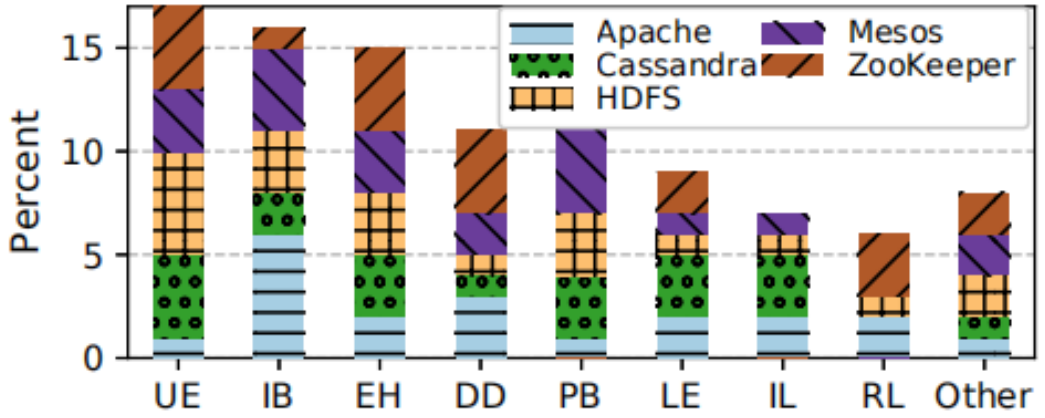
发现1：百分之54的局部故障出现在近三年的软件发行版本中。

原因：随着软件不断添加新的功能和优化，语义变得越来越复杂，一些功能可能会影响到其他功能。例如：HDFS在版本0.23中添加了一个short-circuit本地读取的功能，为了实现此功能需要添加一个DomainSocketWatcher监视Unix域端口，并当端口变得可读时进行回调。但是这个新的模块可能会在运行时意外退出并使执行short-circuit读取的应用卡住。

2.2 研究发现

发现2：造成局部故障的原因是多种多样的。

并没有一个主要或者统一的原因。前三种原因是uncaught errors,indefinite blocking, and buggy error handling，合起来占了48%。



Root cause distribution.

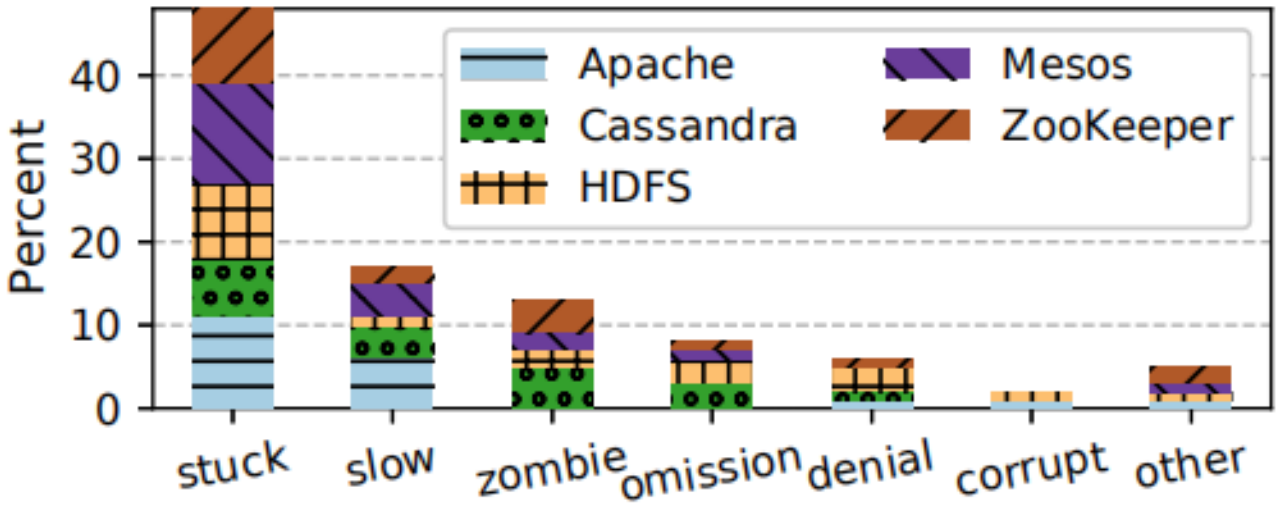
UE: uncaught error; IB: indefinite blocking; EH: buggy error handling;
DD: deadlock; PB: performance bug; LE: logic error; IL: infinite loop; RL: resource leak.

2.2 研究发现

发现3：48%的局部故障造成某个功能卡住。17%的局部故障造成某个操作花费很长时间去完成。

发现4：局部故障中有13%，一个模块变成了具有未定义故障语义的zombie。

- 这通常发生在故障模块意外退出了他的循环，或者即使遇到严重的错误仍然继续执行。



Consequence of studied failures.

2.2 研究发现

发现5：15%的局部故障是静默的。（包括数据丢失，不一致，错误的结果）

- 例如在一个案例中，因为后台连接的错误关闭，且未报告，Apache web服务器在客户请求.js文件时返回了image文件

发现6：71%的局部故障是由特定的生产环境或者其他进程的故障触发的。(错误输入，调度，资源争用，磁盘碎片)

- 例如Zookeeper中的一个局部故障只会当损坏的信息出现在一条记录的length字段才会触发。

2.2 研究发现

发现7：68%的局部故障是粘性的（sticky），剩下32%的局部故障是暂时的（transient）

- sticky意思是故障进程不能自己修复，需要维护人员修复。transient意思是当某个环境条件改变时故障进程可能会复原。

发现8：这些局部故障的平均诊断时间是6天5小时。

- 虽然有些故障原因是很简单的但是诊断仍然花费了很长时间，一是因为局部故障的特征具有迷惑性，误导了诊断方向；二是因为故障进程的错误信息暴露得太少了

2.3 启示

1.局部故障不仅普遍存在于大型系统软件中而且很难诊断和处理。其中68%的故障都不会自己修复，需要维护人员的帮助。

2.大多数局部故障是production-dependent，意味着需要系统运行起来才能检测，所以需要runtime mechanism。

3.如果一个运行时检测器不仅能检测故障还能定位故障，无疑是极好的。

4.让开发人员手动的在代码中添加检测逻辑，比如为每个函数添加try catch，是不现实的。所以需要系统地自动生成检测程序。根据发现3，大多数故障会破坏活性或触发显示的错误，所以在不需要深刻的语义理解的条件下可以自动构造检测器。

03.

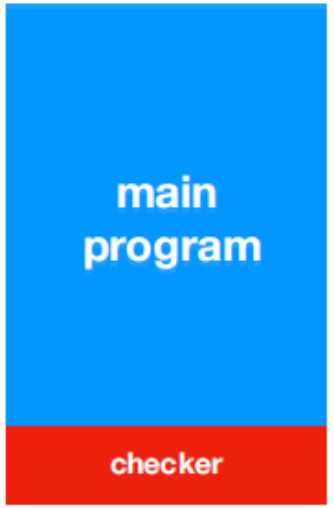
用看门狗捕获局部故障

Catching Partial Failures with Watchdogs

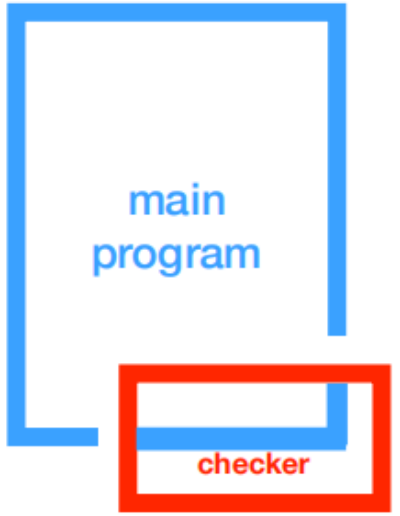
3.1 设计原则

原则：checker应该与main program相交

- 现有检测器中的checker：例如， heartbeat and HTTP tests，与main program不相交，所以无效。
- watchdog中的checker遵循了上面的原则，所以有效。



existing approach



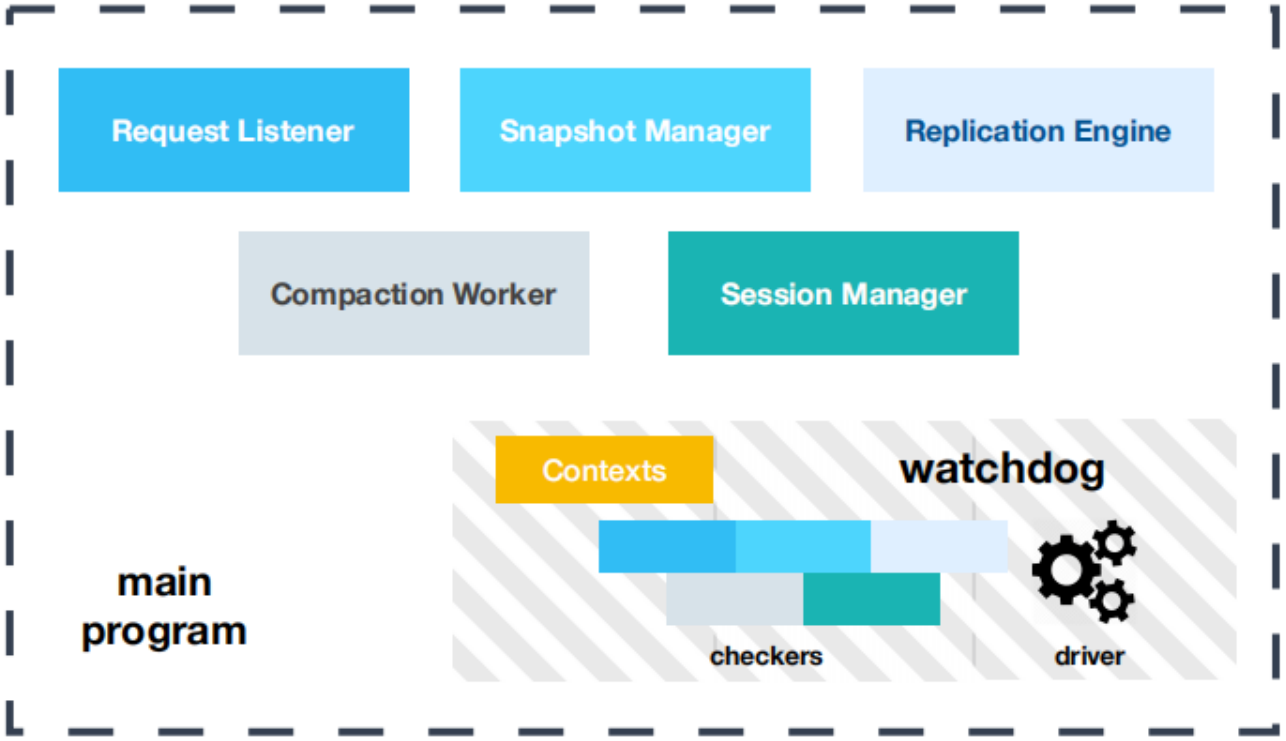
our approach

3.2 核心思想

检测的关键目标：让watchdog经历和主程序类似的故障。

所以，核心思想就是 mimic checker。

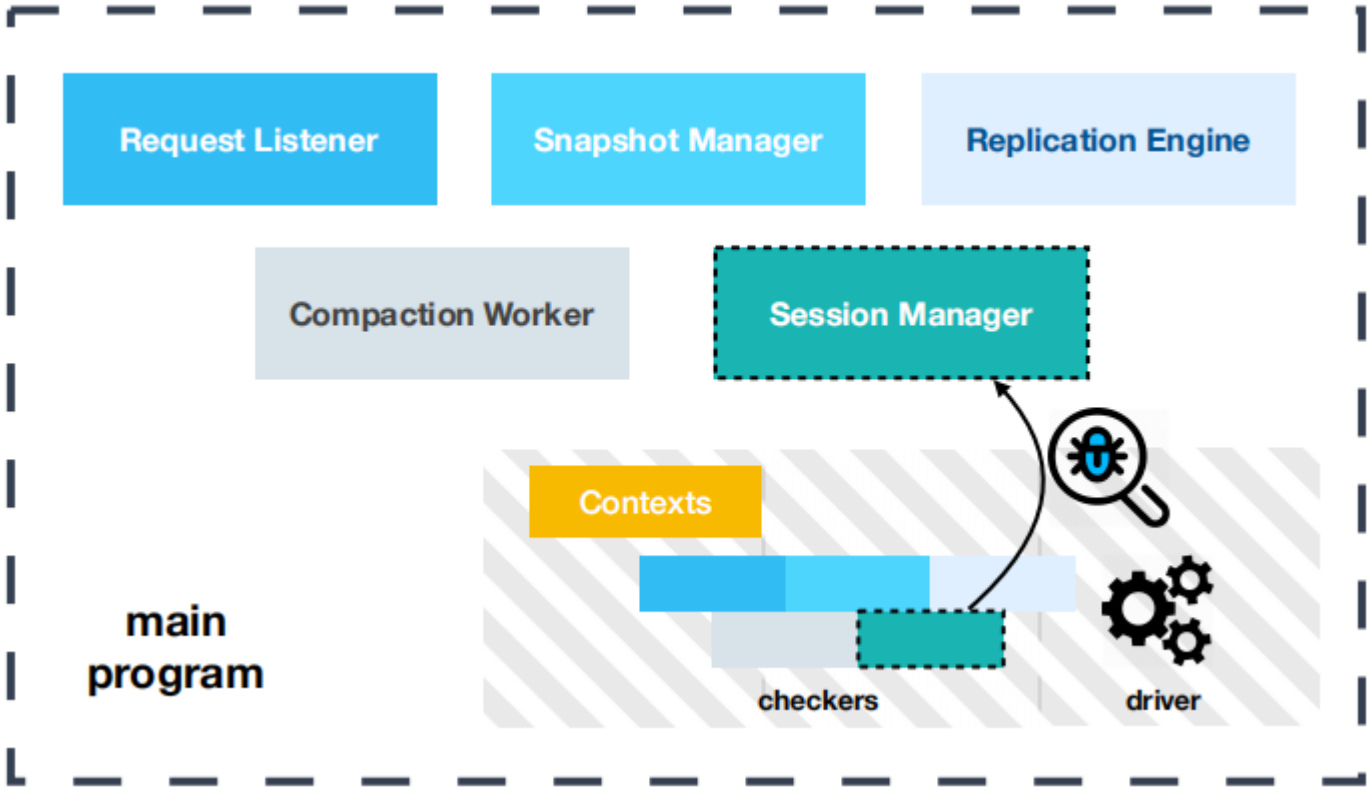
- 该checker从主程序的每个模块中选择一些代表性的操作，模仿他们并检测故障。由于和主程序代码逻辑类似，所以能精确反应主程序的状态，并精准找到故障位置。



3.3 watchdog三大特征

特征一：customized

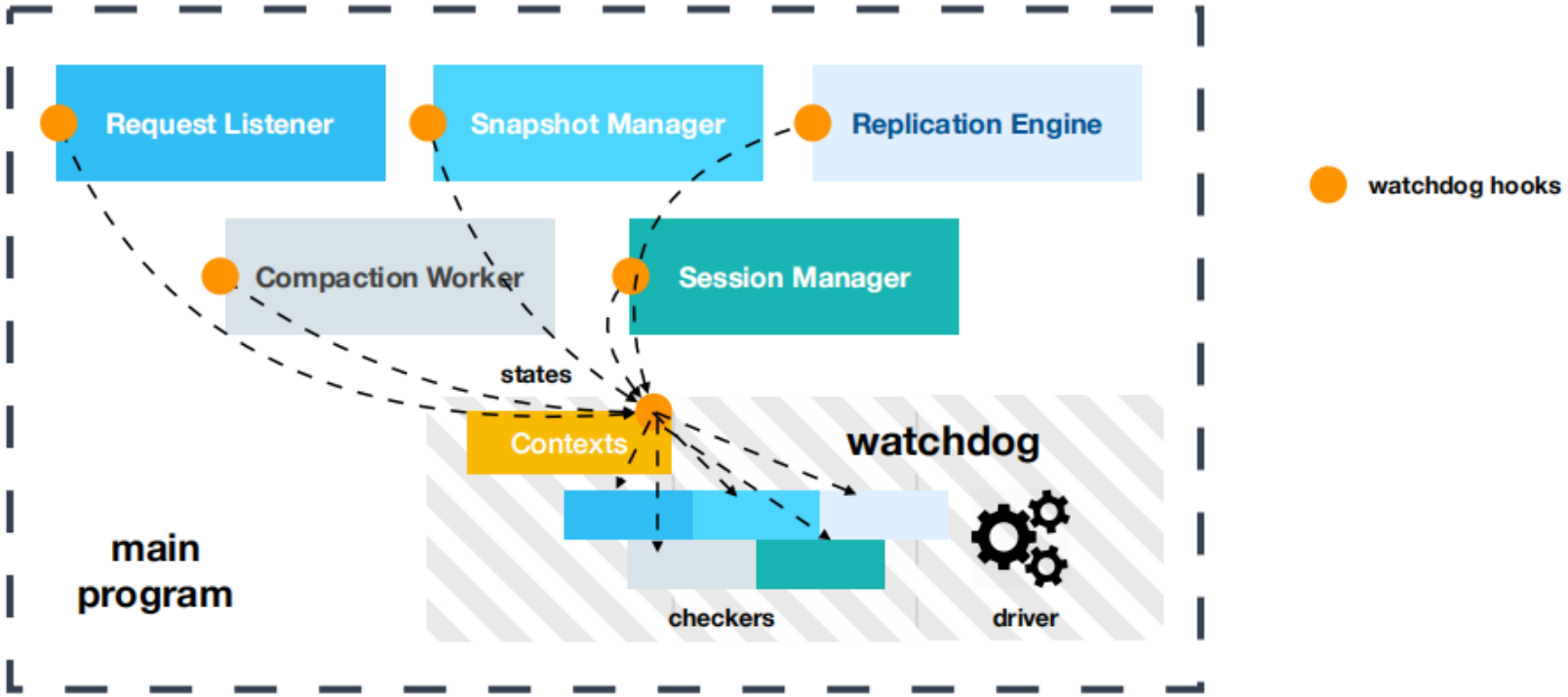
- watchdog中每个checker都是为主程序中的某个模块定制的，它专门用来检查该模块的故障
- watchdog中的driver管理checker的调度和执行



3.3 watchdog三大特征

特征二：stateful

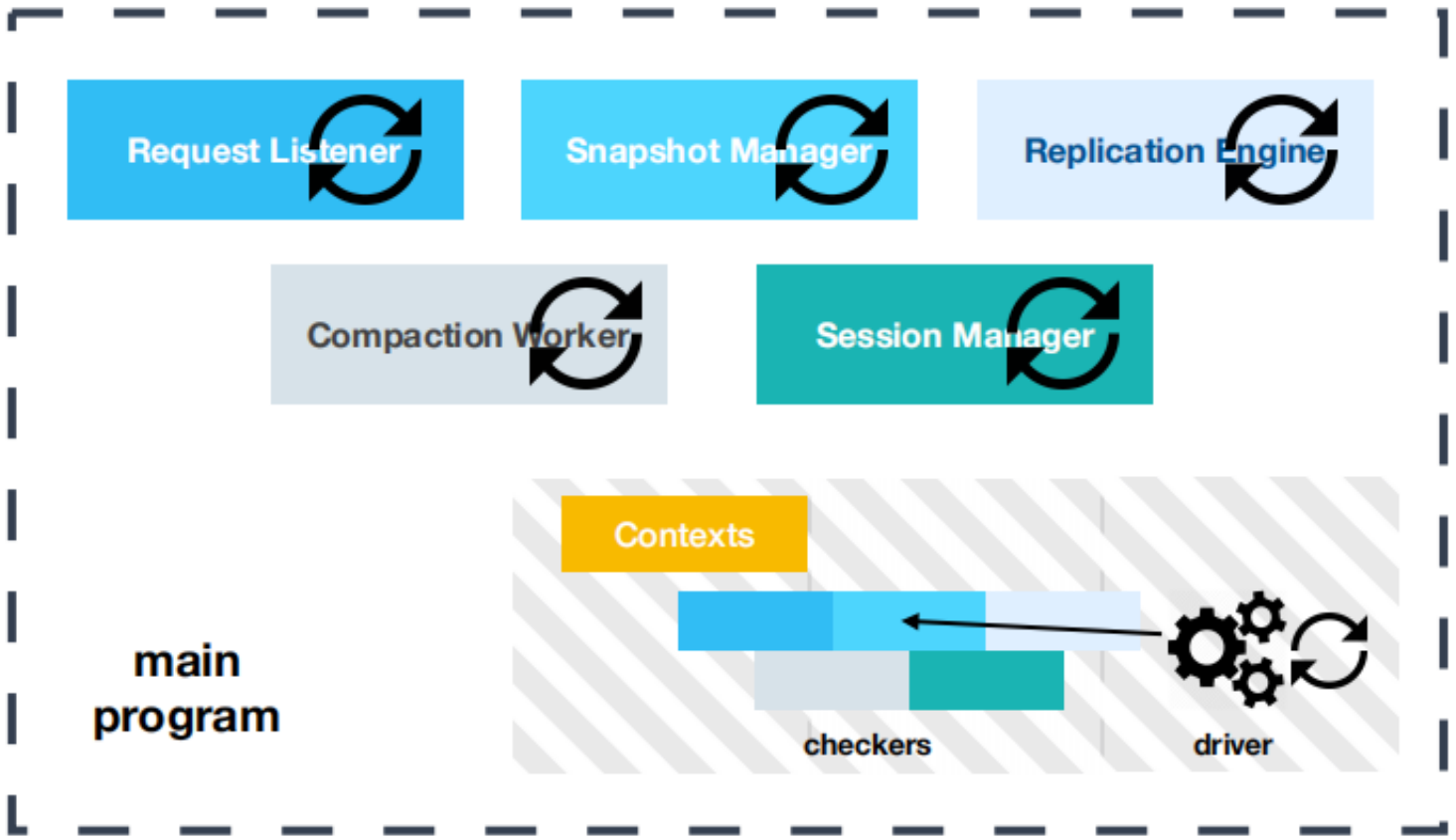
- 在checker进行检测过程中需要使用主程序的最新的状态。这个状态被称作context。
- context在watchdog，它通过hooks与主程序的状态进行同步，当主程序执行到某个hook point时，该hook使用当前程序状态更新context。



3.3 watchdog三大特征

特征三：concurrent

- checker与主程序并行运行。前人是在主程序中添加大量的checker，这样大大影响了主程序的性能，而并行运行的话watchdog不会影响主程序性能。



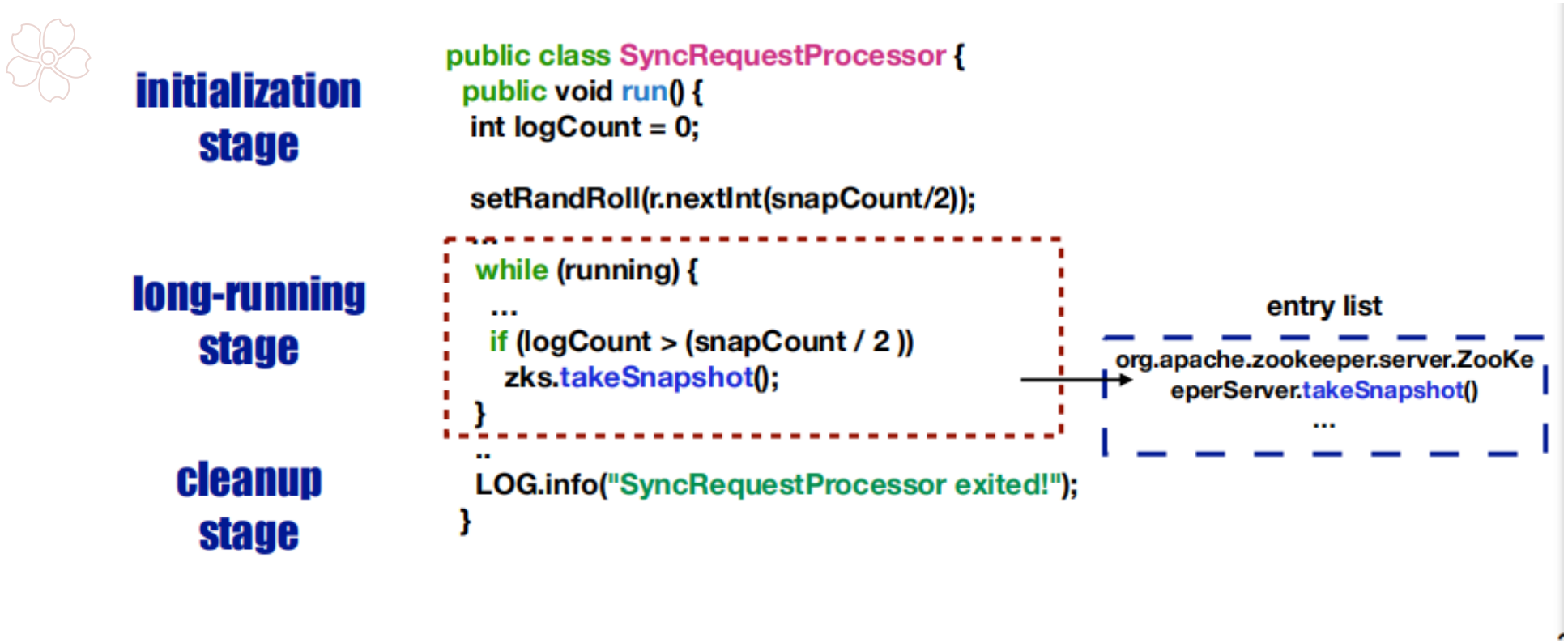
04.

用 OmegaGen 生成看门狗

Generating Watchdogs with OmegaGen

4.1 找到长时间运行的方法

OmegaGen首先找到程序中长时间运行的方法，比如while（true）或 while（flag），有很多一次性执行的任务则被排除，比如创建数据库的操作。



4.2 找到易出错的操作

OmegaGen然后在找到的长时间运行的方法中进一步找到易出错的操作（Vulnerable Operations）

- OmegaGen使用启发式的方法，启发式方法认为synchronization，resource allocation，event polling，async waiting，invocation with external input argument, I/O等操作是vulnerable。
- OmegaGen识别他们通过标准库函数调用，比如oa.writeRecord()因为执行了写操作，所以被认为vulnerable。
- 开发人员也可以在代码中通过@vulnerable标签来注释，这样OmegaGen也可以识别。

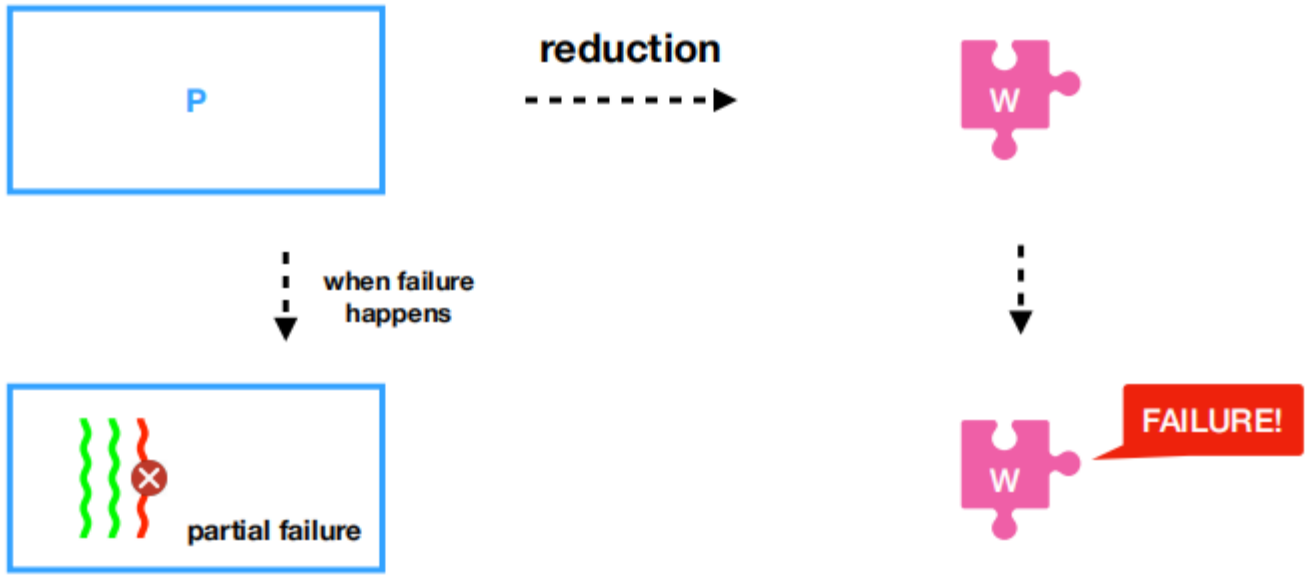
为什么这些操作被认为vulnerable，并且要找到他们呢？

答：根据前面的发现7，大多数局部故障是由运行时特定的运行环境触发的，所以严重受运行环境影响的操作更可能出故障。像有些操作的正确与否是逻辑上决定的，比如排序算法，可通过静态分析找到，交给watchdog就没有必要。

4.3 简化主程序

什么是程序简化？

- 给定一个程序P，简化程序P生成watchdog W，W能够检测P中的局部故障且不影响P的运行



4.3 简化主程序

为什么要进行程序简化？

- 如果将所有代码段放入checker中，checker可能会超时，并且无法定位故障
- 正如4.2所说，我们只关注vulnerable operation，像是初始化阶段的代码，cleanup阶段的代码，逻辑上可以判断正确性的代码都可以简化掉。

4.3 简化主程序

怎样进行程序简化？

- 因为前两步工作标识好了长时间运行的方法和方法里的vulnerable operation，然后从上而下地将非vulnerable operation删除掉，如果整个方法里不存在vulnerable operation则将这个方法删除掉。
- 如果在一个方法里某个vulnerable operation已经出现过，则后面再次出现的vulnerable operation也会被删除，因为后面的在暴露故障时是多余的。

4.4 封装简化后的程序

OmegaGen将简化后的代码段封装进watchdog中，但是这些代码段是不能直接执行的。

- OmegaGen分析简化后方法执行所需要的所有的参数，然后为每个未定义的变量添加一个本地变量定义
- OmegaGen进一步生成了一个context factory，提供API管理简化方法的所有参数

```
public static void serializeNode_invoke() {
  Context ctx = ContextManger.
      serializeNode_reduced_context();
  if (ctx.status == READY) {
    OutputArchive arg0 = ctx.args_getter(0);
    DataNode arg1 = ctx.args_getter(1);
    serializeNode_reduced(arg0, arg1);
  }
}
```

4 generate context factory

4.5 插入context hooks

在主程序中，vulnerable operation的前面插入context hook，hook能够调用context factory的setter方法来同步状态

```
void serializeNode(OutputArchive oa, StringBuilder path) throws IOException {
    String pathString = path.toString();
    DataNode node = getNode(pathString);
+   ContextFactory.serializeNode_context_setter(oa, node);
    String children[] = null;
    synchronized (node) {
        oa.writeRecord(node, "node");
        Set<String> childs = node.getChildren();
        if (childs != null)
            children = childs.toArray(new String[childs.size()]);
    }
    path.append('/');
    int off = path.length();
    ...
}
```

insert context hook before vulnerable operation

4.6 预防副作用

watchdog和主程序并行运行可能会对主程序造成影响

- memory side effect：watchdog checker可能会意外修改主程序的状态

  解决办法：复制context。这样能确保任何修改都只保存在watchdog的状态中
- I/O side effect：watchdog和主程序可能同时对同一文件进行写操作，会影响主程序接下来的执行

  解决办法：OmegaGen在watchdog中添加了I/O重定向功能。当发生访存冲突时，让watchdog重定向到其他文件。

05.

评估结果

Evaluation Results

5.1 生成的watdchdog

请注意：这些watchdog是静态的watchdog，其中只有一部分才会在运行时被激活。

	ZooKeeper	Cassandra	HDFS	HBase	MapReduce	Yarn
Watchdogs	96	190	174	358	161	88
Methods	118	464	482	795	371	222
Operations	488	2,112	3,416	9,557	6,116	752

Number of watchdogs and checkers generated

5.2 真实的局部故障

为了评估生成的watchdog的有效性，我们收集和重现了6个分布式系统的22个真实世界的局部故障

JIRA Id.	Id.	Root Cause	Conseq.	Sticky?	Study?
ZooKeeper-2201	ZK1	Bad Synch.	Stuck	No	Yes
ZooKeeper-602	ZK2	Uncaught Error	Zombie	Yes	Yes
ZooKeeper-2325	ZK3	Logic Error	Inconsist	Yes	No
ZooKeeper-3131	ZK4	Resource Leak	Slow	Yes	Yes
Cassandra-6364	CS1	Uncaught Error	Zombie	Yes	Yes
Cassandra-6415	CS2	Indefinite Blocking	Stuck	No	Yes
Cassandra-9549	CS3	Resource Leak	Slow	Yes	No
Cassandra-9486	CS4	Performance Bug	Slow	Yes	No
HDFS-8429	HF1	Uncaught Error	Stuck	Yes	Yes
HDFS-11377	HF2	Indefinite Blocking	Stuck	No	Yes
HDFS-11352	HF3	Deadlock	Stuck	Yes	No
HDFS-4233	HF4	Uncaught Error	Data Loss	Yes	No
HBase-18137	HB1	Infinite Loop	Stuck	Yes	No
HBase-16429	HB2	Deadlock	Stuck	Yes	No
HBase-21464	HB3	Logic Error	Stuck	Yes	No
HBase-21357	HB4	Uncaught Error	Denial	Yes	No
HBase-16081	HB5	Indefinite Blocking	Silent	Yes	No
MapReduce-6351	MR1	Deadlock	Stuck	Yes	No
MapReduce-6190	MR2	Infinite Loop	Stuck	Yes	No
MapReduce-6957	MR3	Improper Err Handling	Stuck	Yes	No
MapReduce-3634	MR4	Uncaught Error	Zombie	Yes	No
Yarn-4254	YN1	Improper Err Handling	Stuck	Yes	No

5.3 检测时间结果

- 检测时间是用watchdog第一次报告故障的时间减去程序到达故障点的时间计算的。
- watchdog能够检测到20个故障，平均检测时间是4.2秒
- watchdog在检测liveness issue（如死锁，无限阻塞）和safety issue（如exception）是有效的，但是检测silent是无效的
- 作比较的四个baseline detector，总共才检测出14个故障

	ZK1	ZK2	ZK3	ZK4	CS1	CS2	CS3	CS4	HF1	HF2	HF3	HF4	HB1	HB2	HB3	HB4	HB5	MR1	MR2	MR3	MR4	YN1
Client	✗	2.47	2.27	✗	441	✗	✗	✗	✗	✗	✗	✗	✗	4.81	✗	6.62	✗	✗	✗	✗	8.54	7.38
Probe	✗	✗	✗	✗	15.84	✗	✗	✗	✗	✗	✗	✗	✗	4.71	✗	7.76	✗	✗	✗	✗	✗	✗
Signal	12.2	0.63	1.59	0.4	5.31	✗	✗	✗	✗	✗	✗	0.77	0.619	✗	0.62	61.0	✗	✗	✗	✗	0.60	1.16
Res.	5.33	0.56	0.72	17.17	209.5	✗	-19.65	✗	-3.13	✗	✗	0.83	✗	✗	✗	0.60	✗	✗	✗	✗	✗	✗
Watch	4.28	-5.89	3.00	41.19	-3.73	4.63	46.56	38.72	1.10	6.20	3.17	2.11	5.41	7.89	✗	0.80	5.89	1.01	4.07	1.46	4.68	✗

Detection times (in secs) for the real-world cases

比较Panorama 和OmegaGen

Panorama将分布式系统的各个组件转化为observer，通过组件间的交互生成观察结果来检测局部故障。

- requester从外部观察到的故障具有局限性，对于进程内部的故障毫无办法

OmegaGen通过静态分析main program的源码，从main program派生出简化版本的watchdog，通过在生产环境中测试watchdog来检测局部故障。

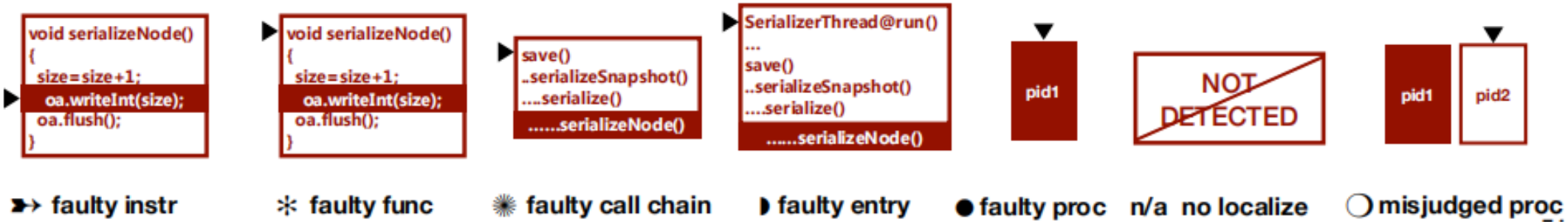
- watchdog是从main program中派生出来的，更准确的反应主程序的状态
- watchdog非常精简，方便精准定位故障位置

5.4 检测位置结果

将检测位置的精确度度分为了六个等级。watchdog能够找到11个局部故障的出错的指令，7个局部故障的出错函数或函数调用链，这证明watchdog是有效的。

	ZK1	ZK2	ZK3	ZK4	CS1	CS2	CS3	CS4	HF1	HF2	HF3	HF4	HB1	HB2	HB3	HB4	HB5	MR1	MR2	MR3	MR4	YN1
Client	n/a	●	●	n/a	●	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	●	n/a	○	n/a	n/a	n/a	n/a	●	●
Probe	n/a	n/a	n/a	n/a	◗	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	◗	n/a	◗	n/a	n/a	n/a	n/a	n/a	n/a
Signal	●	➳	●	●	➳	n/a	n/a	n/a	n/a	n/a	n/a	➳	➳	n/a	✺	✺	n/a	n/a	n/a	n/a	➳	➳
Res.	●	●	●	●	●	n/a	●	n/a	●	n/a	n/a	●	n/a	n/a	n/a	●	n/a	n/a	n/a	n/a	n/a	n/a
Watch	➳	➳	●	✻	➳	✻	●	✻	✻	✺	➳	➳	➳	➳	n/a	➳	✺	➳	➳	✺	➳	n/a

Failure localization for the real-world cases.



5.5 误报结果

watchdog报告的故障可能是暂时的或者可以容忍的，所以会存在误报的情况，当引入验证机制后（watch_v），相比于原有版本（watch），误报的比例明显下降

	ZooKeeper	Cassandra	HDFS	HBase	MapReduce	Yarn
probe	0%	0%	0%	0%	0%	0%
resource	0%-3.4%	0%-6.3%	0.05%-3.5%	0%-3.72%	0.33%-0.67%	0%-6.1%
signal	3.2%-9.6%	0%	0%-0.05%	0%-0.67%	0%	0%
watch.	0%-0.73%	0%-1.2%	0%	0%-0.39%	0%	0%-0.31%
watch_v.	0%-0.01%	0%	0%	0%-0.07%	0%	0%

False alarm ratios of all detectors for six systems under various setups

5.6 静态分析性能结果

OmegaGen静态分析生成watchdog要经过分析和生成两个阶段，在评估的六个分布式系统中除了Hbase，其他五个整个过程的时间都少于5分钟。因为Hbase代码库是最大的。

	ZK	CS	HF	HB	MR	YN
Analysis	21	166	75	92	55	50
Generation	43	103	130	953	131	89

Table 8: OmegaGen watchdog generation time (sec).

5.7 系统开销结果

watchdog会造成5.0%-6.6%的吞吐量上的开销。主要的开销来自于watchdog hooks而不是并行执行checker。

	ZK	CS	HF	HB	MR	YN
Base	428.0	3174.9	90.6	387.1	45.0	45.0
w/ Watch.	399.8	3014.7	85.1	366.4	42.1	42.3
w/ Probe.	417.6	3128.2	89.4	374.3	44.9	44.9
w/ Resource.	424.8	3145.4	89.9	385.6	44.9	44.6

Table 9: System throughput (op/s) w/ different detectors.

06.

缺陷和不足

Limitation and Shortcoming

- 在找vulnerable operation时使用启发式方法，这一步可以通过离线分析或动态自适应选择进行改进
- watchdog在找liveness issue（如死锁，无限阻塞）和safety issue（如exception）是有效的，在找silent语义故障时是无效的
- watchdog只能报告单个进程的故障，一个改进是将OmegaGen和 failure detector整合到一起，这样一个进程的detector能够检查另一个进程的watchdog
- watchdog只能检测和定位故障不能修复故障
- watchdog只能检测runtime error不能检测logical error。因为在程序简化阶段只留下了vulnerable operation，逻辑上可以检测出错误的代码都删除了。

感谢聆听