CrashTuner: Detecting Crash Recovery Bugs in Cloud Systems via Meta-info Analysis

SOSP2019

Jie Lu, Chen Liu, Lian Li, Xiaobing Feng
Feng Tan, Jun Yang, Liang You

Institute of Computing Technology, Chinese Academy of Sciences

University of Chinese Academy of Sciences. Alibaba Group

Crash Recovery

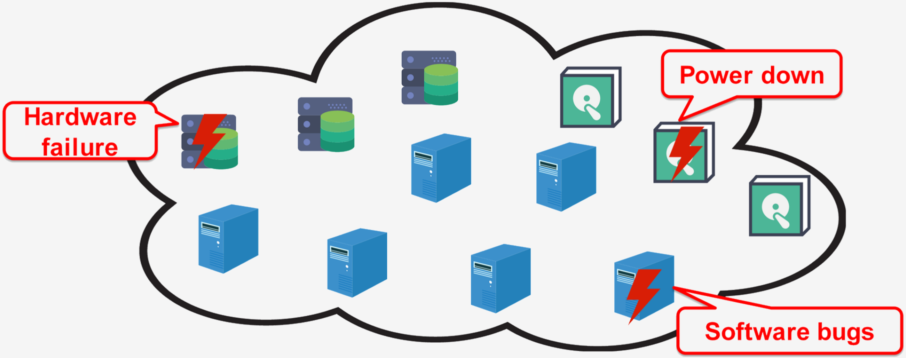
- Recovery must be a first-class operation of distributed systems[1].

[1] Brian F Cooper et al. (2010). "Benchmarking cloud serving systems with YCSB". In: *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, pp. 143–154.

[2] Haryadi S Gunawi et al. (2014). "What bugs live in the cloud? a study of 3000+ issues in cloud systems". In: *Proceedings of the ACM Symposium on Cloud*

Crash Recovery

- Recovery must be a first-class operation of distributed systems[1].
  - Nodes can crash due to different reasons.[2]

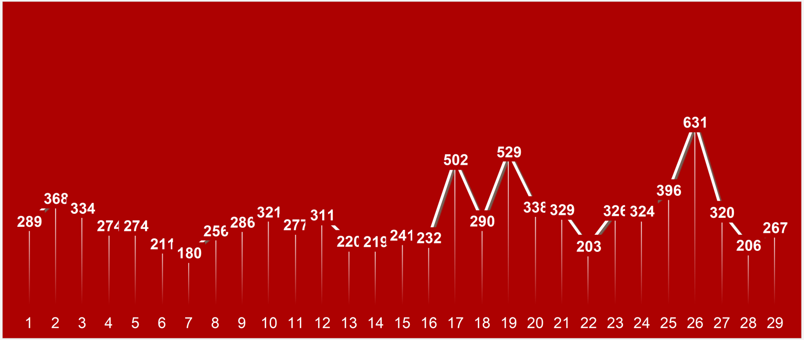


[1] Brian F Cooper et al. (2010). "Benchmarking cloud serving systems with YCSB". In: *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, pp. 143–154.

[2] Haryadi S Gunawi et al. (2014). "What bugs live in the cloud? a study of 3000+ issues in cloud systems". In: *Proceedings of the ACM Symposium on Cloud*

Crash Recovery

- Recovery must be a first-class operation of distributed systems[3].
  - Node Crash Events can be common in a large cluster(At least 180).[4]



[3] Brian F Cooper et al. (2010). "Benchmarking cloud serving systems with YCSB". In: *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, pp. 143–154.

[4] Mohammad Reza Mesbahi, Amir Masoud Rahmani, and Mehdi Hosseinzadeh (2017). "Cloud dependability analysis: Characterizing google cluster infrastructure reliability". In: *2017 3th International Conference on Web Research (ICWR)*. IEEE, pp. 56–61.

Crash-Recovery Bugs and Detection

- Crash Recovery Code can be buggy and often result in catastrophic failure.[5]

[5] Haryadi S Gunawi et al. (2014). "What bugs live in the cloud? a study of 3000+ issues in cloud systems". In: *Proceedings of the ACM Symposium on Cloud Computing*. ACM, pp. 1–14.

[6] Haopeng Liu et al. (2018). "Fcatch: Automatically detecting time-of-fault bugs in cloud systems". In: *ACM SIGPLAN Notices* 53.2, pp. 419–431.

[7] Tanakorn Leesatapornwongsa et al. (2014). "{SAMC}: Semantic-Aware Model Checking for Fast Discovery of Deep Bugs in Cloud Systems". In: *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pp. 399–414.

[8] Yu Gao et al. (2018). "An Empirical Study on Crash Recovery Bugs in Large-Scale Distributed Systems". In: *Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2018.

Crash-Recovery Bugs and Detection

- Crash Recovery Code can be buggy and often result in catastrophic failure.[5]
- Existing detection approaches

[5] Haryadi S Gunawi et al. (2014). "What bugs live in the cloud? a study of 3000+ issues in cloud systems". In: *Proceedings of the ACM Symposium on Cloud Computing*. ACM, pp. 1–14.

[6] Haopeng Liu et al. (2018). "Fcatch: Automatically detecting time-of-fault bugs in cloud systems". In: *ACM SIGPLAN Notices* 53.2, pp. 419–431.

[7] Tanakorn Leesatapornwongsa et al. (2014). "{SAMC}: Semantic-Aware Model Checking for Fast Discovery of Deep Bugs in Cloud Systems". In: *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pp. 399–414.

[8] Yu Gao et al. (2018). "An Empirical Study on Crash Recovery Bugs in Large-Scale Distributed Systems". In: *Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2018.

Crash-Recovery Bugs and Detection

- Crash Recovery Code can be buggy and often result in catastrophic failure.[5]
- Existing detection approaches
  - Random fault injection: Ineffective[6].

[5] Haryadi S Gunawi et al. (2014). "What bugs live in the cloud? a study of 3000+ issues in cloud systems". In: *Proceedings of the ACM Symposium on Cloud Computing*. ACM, pp. 1–14.

[6] Haopeng Liu et al. (2018). "Fcatch: Automatically detecting time-of-fault bugs in cloud systems". In: *ACM SIGPLAN Notices* 53.2, pp. 419–431.

[7] Tanakorn Leesatapornwongsa et al. (2014). "{SAMC}: Semantic-Aware Model Checking for Fast Discovery of Deep Bugs in Cloud Systems". In: *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pp. 399–414.

[8] Yu Gao et al. (2018). "An Empirical Study on Crash Recovery Bugs in Large-Scale Distributed Systems". In: *Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2018.

Crash-Recovery Bugs and Detection

- Crash Recovery Code can be buggy and often result in catastrophic failure.[5]
- Existing detection approaches
  - Random fault injection: Ineffective[6].
  - Model checking: Inefficient and requires manual specifications[7].

---

[5] Haryadi S Gunawi et al. (2014). "What bugs live in the cloud? a study of 3000+ issues in cloud systems". In: *Proceedings of the ACM Symposium on Cloud Computing*. ACM, pp. 1–14.

[6] Haopeng Liu et al. (2018). "Fcatch: Automatically detecting time-of-fault bugs in cloud systems". In: *ACM SIGPLAN Notices* 53.2, pp. 419–431.

[7] Tanakorn Leesatapornwongsa et al. (2014). "{SAMC}: Semantic-Aware Model Checking for Fast Discovery of Deep Bugs in Cloud Systems". In: *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pp. 399–414.

[8] Yu Gao et al. (2018). "An Empirical Study on Crash Recovery Bugs in Large-Scale Distributed Systems". In: *Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2018.

Crash-Recovery Bugs and Detection

- Crash Recovery Code can be buggy and often result in catastrophic failure.[5]
- Existing detection approaches
  - Random fault injection: Ineffective[6].
  - Model checking: Inefficient and requires manual specifications[7].
- Crash-Recovery bugs still widely exist in distributed system.[8]

---

[5] Haryadi S Gunawi et al. (2014). "What bugs live in the cloud? a study of 3000+ issues in cloud systems". In: *Proceedings of the ACM Symposium on Cloud Computing*. ACM, pp. 1–14.

[6] Haopeng Liu et al. (2018). "Fcatch: Automatically detecting time-of-fault bugs in cloud systems". In: *ACM SIGPLAN Notices* 53.2, pp. 419–431.

[7] Tanakorn Leesatapornwongsa et al. (2014). "{SAMC}: Semantic-Aware Model Checking for Fast Discovery of Deep Bugs in Cloud Systems". In: *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pp. 399–414.

[8] Yu Gao et al. (2018). "An Empirical Study on Crash Recovery Bugs in Large-Scale Distributed Systems". In: *Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2018.

Crash-Recovery Bugs and Detection

- Crash Recovery Code can be buggy and often result in catastrophic failure.[5]
- Existing detection approaches
  - Random fault injection: Ineffective[6].
  - Model checking: Inefficient and requires manual specifications[7].
- Crash-Recovery bugs still widely exist in distributed system.[8]
  - Distributed systems have large state space to explore.

[5] Haryadi S Gunawi et al. (2014). "What bugs live in the cloud? a study of 3000+ issues in cloud systems". In: *Proceedings of the ACM Symposium on Cloud Computing*. ACM, pp. 1–14.

[6] Haopeng Liu et al. (2018). "Fcatch: Automatically detecting time-of-fault bugs in cloud systems". In: *ACM SIGPLAN Notices* 53.2, pp. 419–431.

[7] Tanakorn Leesatapornwongsa et al. (2014). "{SAMC}: Semantic-Aware Model Checking for Fast Discovery of Deep Bugs in Cloud Systems". In: *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pp. 399–414.

[8] Yu Gao et al. (2018). "An Empirical Study on Crash Recovery Bugs in Large-Scale Distributed Systems". In: *Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2018.

Crash-Recovery Bugs and Detection

- Crash Recovery Code can be buggy and often result in catastrophic failure.[5]
- Existing detection approaches
  - Random fault injection: Ineffective[6].
  - Model checking: Inefficient and requires manual specifications[7].
- Crash-Recovery bugs still widely exist in distributed system.[8]
  - Distributed systems have large state space to explore.
  - Crash-Recovery bugs can only be triggered when nodes crash under special timing conditions.

[5] Haryadi S Gunawi et al. (2014). "What bugs live in the cloud? a study of 3000+ issues in cloud systems". In: *Proceedings of the ACM Symposium on Cloud Computing*. ACM, pp. 1–14.

[6] Haopeng Liu et al. (2018). "Fcatch: Automatically detecting time-of-fault bugs in cloud systems". In: *ACM SIGPLAN Notices* 53.2, pp. 419–431.

[7] Tanakorn Leesatapornwongsa et al. (2014). "{SAMC}: Semantic-Aware Model Checking for Fast Discovery of Deep Bugs in Cloud Systems". In: *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pp. 399–414.

[8] Yu Gao et al. (2018). "An Empirical Study on Crash Recovery Bugs in Large-Scale Distributed Systems". In: *Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2018.

This paper: CrashTuner

- A new approach to automatically detect crash-recovery bugs in distributed systems .
  - 21 new crash-recovery bugs (including 10 critical bugs).
  - Test 5 distributed systems in 35 hours.

Bug ID	Priority	Scenario	Status	Symptom	Meta-info
YARN-9238	Critical	pre-read	Fixed	Allocating containers to removed ApplicationAttempt	ApplicationAttemptId
YARN-9165	Critical	pre-read	Fixed	Scheduling the removed container	ContainerId
YARN-9193	Critical	pre-read	Fixed	Allocating container to removed node	NodeId
YARN-9164(2)	Critical	pre-read	Fixed	Cluster down due to using the removed node	NodeId
YARN-9201	Major	pre-read	Fixed	Invalid event for current state of ApplicationAttempt	ContainerId
HDFS-14216(2)	Critical	pre-read	Fixed	Request fails due to removed node	DataNodeInfo
YARN-9194	Critical	pre-read	Fixed	Invalid event for current state of ApplicationAttempt	ApplicationId
HBASE-22041	Critical	post-write	Unresolved	Master startup node hang	ServerName
HBASE-22017	Critical	pre-read	Fixed	Master fails to become active due to removed node	ServerName
YARN-8650(2)	Major	pre-read	Fixed	Invalid event for current state of Container	ContainerId
YARN-9248	Major	pre-read	Fixed	Invalid event for current state of Container	ApplicationAttemptId
YARN-8649	Major	pre-read	Fixed	Resource Leak due to removed container	ApplicationId
HBASE-21740	Major	post-write	Fixed	Shutdown during initialization causing abort	MetricsRegionServer
HBASE-22050	Major	pre-read	Unresolved	Atomic violation causing shutdown aborts	RegionInfo
HDFS-14372	Major	pre-read	fixed	Shutdown before register causing abort	BPOfferService
MR-7178	Major	post-write	Unresolved	Shutdown during initialization causing abort	TaskAttemptId
HBASE-22023	Trivial	post-write	Unresolved	Shutdown during initialization causing abort	MetricsRegionServer
CA-15131	Normal	pre-read	Unresolved	Request fails due to using removed node	InetAddressAndPort

The paper: CrashTuner

How does CrashTuner do it?

Findings

- Existing Crash-Recovery bugs can be easily triggered when nodes:

Figure: 116 Crash-Recovery Bugs from four distributed Systems.

hadoop HDFS	hadoop YARN	APACHE HBASE	Apache Zookeeper
Distributed File System	Distributed Resource Manager System	Distributed Database	Centralized service
Master/slave	Master/slave	Master/slave	Master/slave based on leader election

Findings

- Existing Crash-Recovery bugs can be easily triggered when nodes:
  - Crash before reading variables

Figure: 116 Crash-Recovery Bugs from four distributed Systems.

hadoop HDFS	hadoop YARN	APACHE HBASE	Apache Zookeeper
Distributed File System	Distributed Resource Manager System	Distributed Database	Centralized service
Master/slave	Master/slave	Master/slave	Master/slave based on leader election

Findings

- Existing Crash-Recovery bugs can be easily triggered when nodes:
  - Crash before reading variables
  - Crash after writing variables .

Figure: 116 Crash-Recovery Bugs from four distributed Systems.

hadoop HDFS	hadoop YARN	APACHE HBASE	Apache Zookeeper
Distributed File System	Distributed Resource Manager System	Distributed Database	Centralized service
Master/slave	Master/slave	Master/slave	Master/slave based on leader election

Findings

- Existing Crash-Recovery bugs can be easily triggered when nodes:
  - Crash before reading variables
  - Crash after writing variables .
- One thing in common : All these variables are **meta-info** variables.

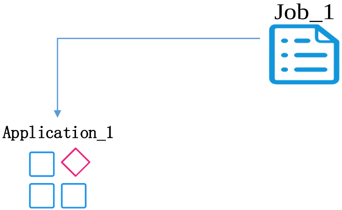
Figure: 116 Crash-Recovery Bugs from four distributed Systems.

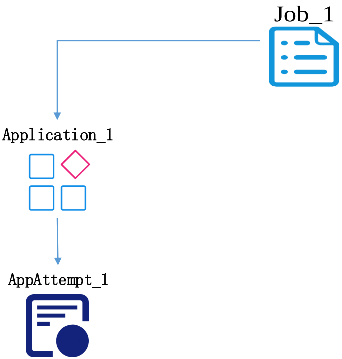
hadoop HDFS	hadoop YARN	APACHE HBASE	Apache Zookeeper
Distributed File System	Distributed Resource Manager System	Distributed Database	Centralized service
Master/slave	Master/slave	Master/slave	Master/slave based on leader election

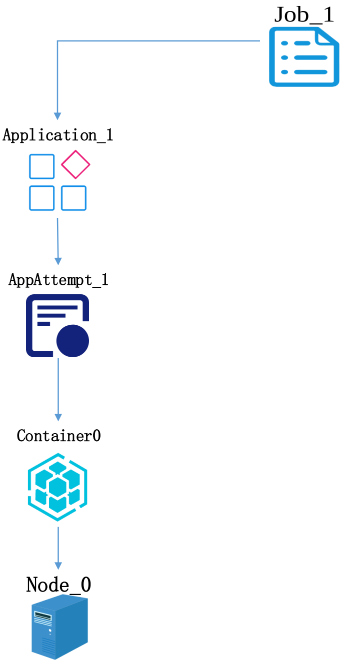
What are meta-info variables?

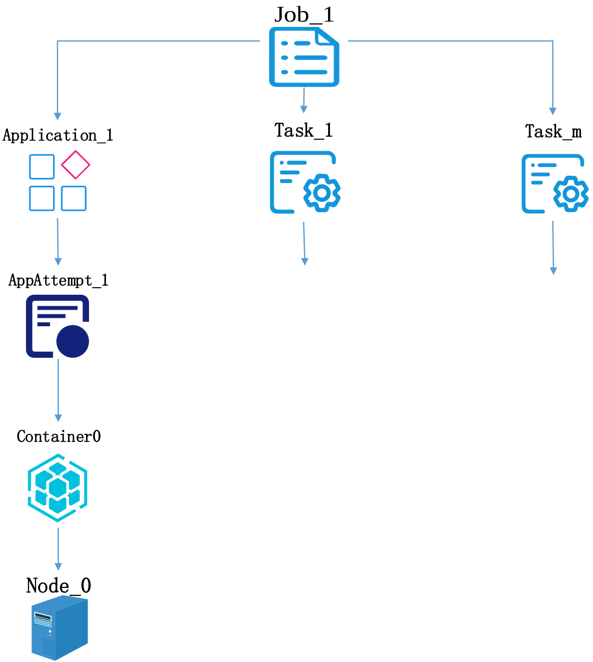
A simplified YARN example

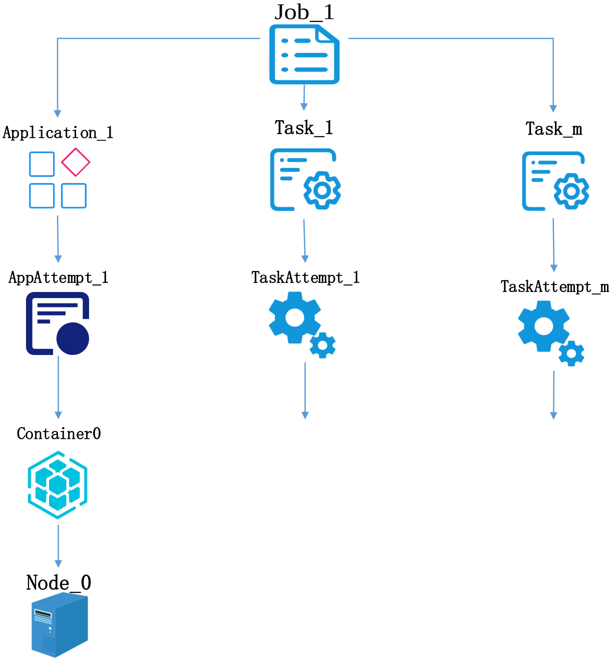
Job_1

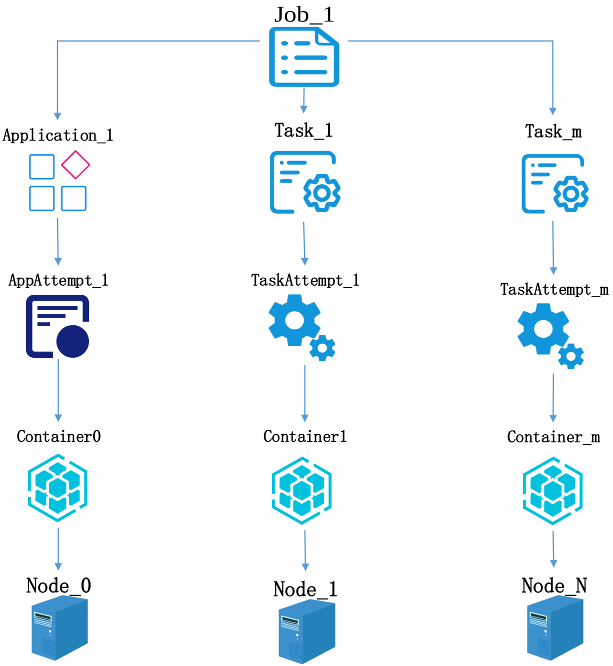

Job_1
Application_1

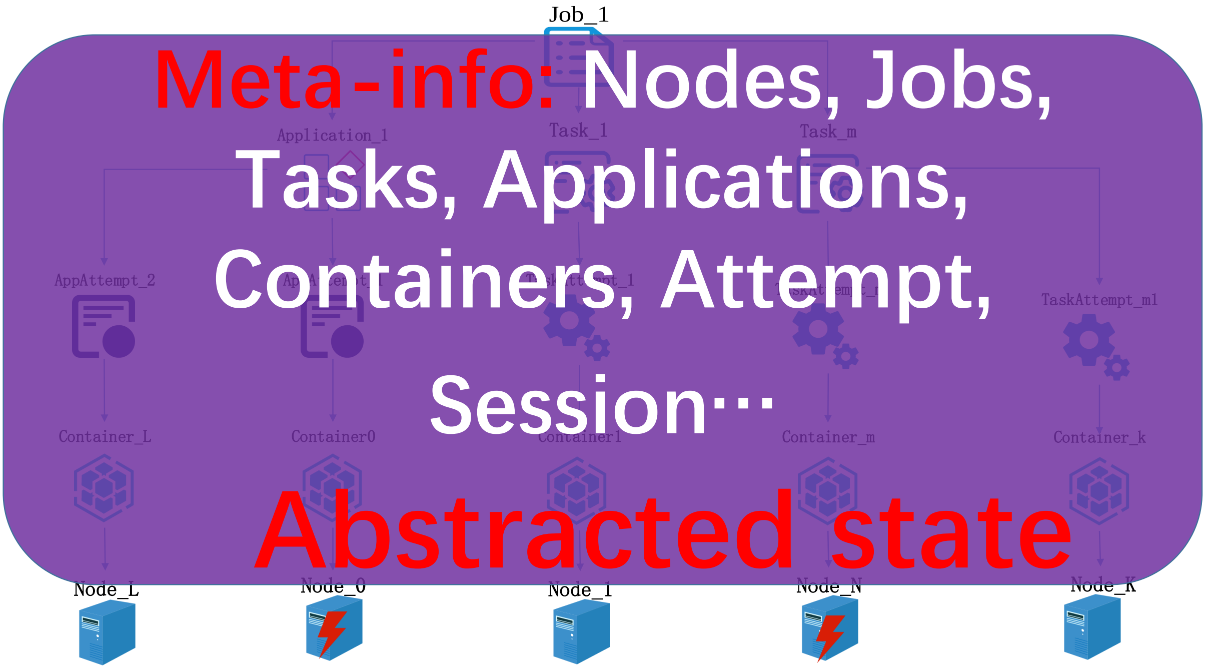

Job_1
Application_1
AppAttempt_1

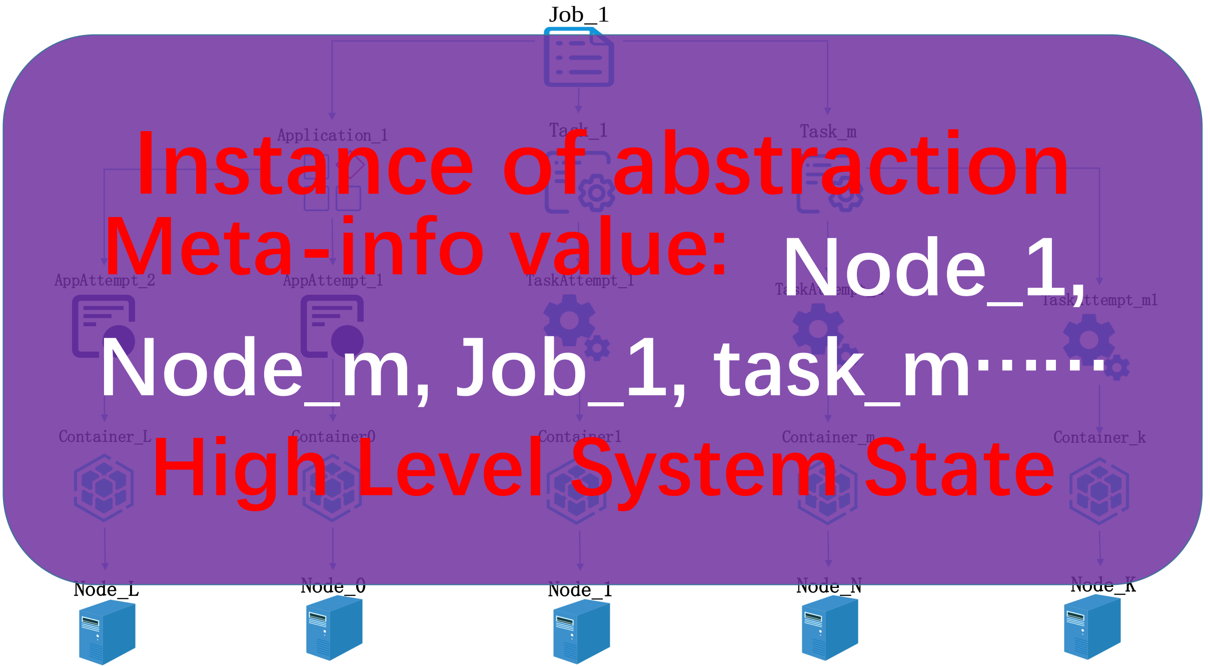

Job_1
Application_1
AppAttempt_1
Container0
Node_0


Job_1
Application_1
Task_1
Task_m
AppAttempt_1
Container0
Node_0

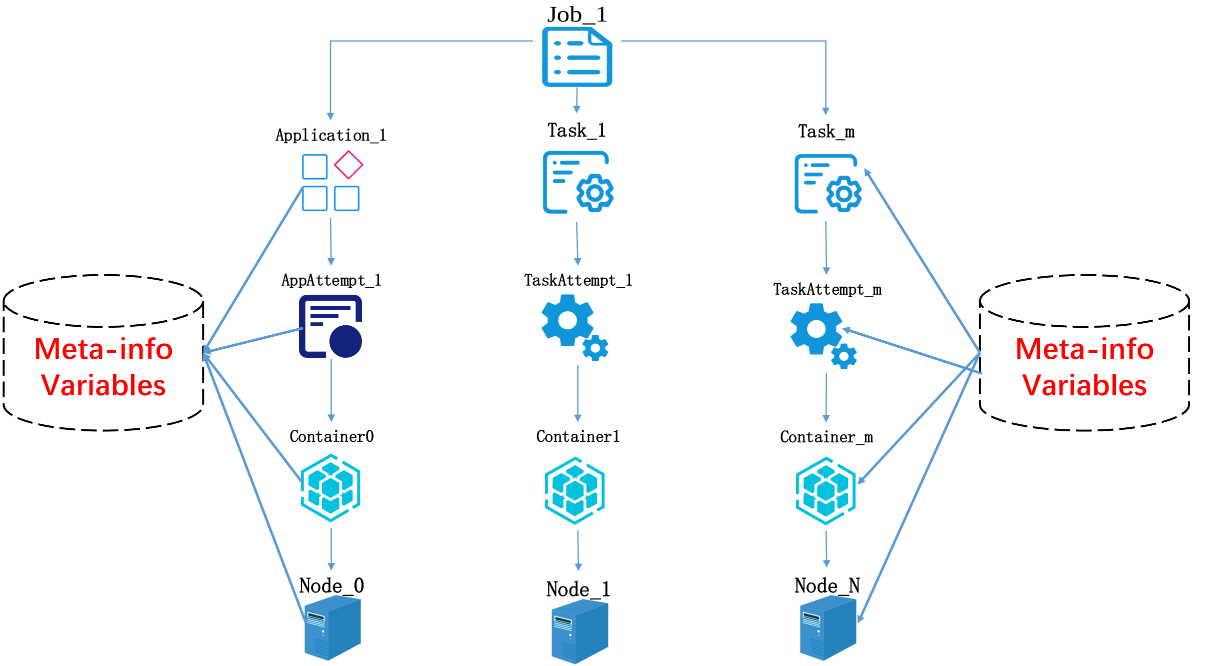

Job_1
Application_1
Task_1
Task_m
AppAttempt_1
TaskAttempt_1
TaskAttempt_m
Container0
Node_0


Job_1
Application_1
Task_1
Task_m
AppAttempt_1
TaskAttempt_1
TaskAttempt_m
Container0
Container1
Container_m
Node_0
Node_1
Node_N


Meta-info: Nodes, Jobs, Tasks, Applications, Containers, Attempt, Session…
Abstracted state
Job_1
Application_1
Task_1
Task_m
AppAttempt_2
TaskAttempt_m1
Container_L
Container0
Container1
Container_m
Container_k
Node_L
Node_0
Node_1
Node_N
Node_K



Job_1
Application_1
Task_1
Task_m
Instance of abstraction
Meta-info value: Node_1, Node_m, Job_1, task_m……
AppAttempt_2
AppAttempt_1
TaskAttempt_1
TaskAttempt_m1
Container_L
Container0
Container1
Container_m
Container_k
High Level System State
Node_L
Node_0
Node_1
Node_N
Node_K

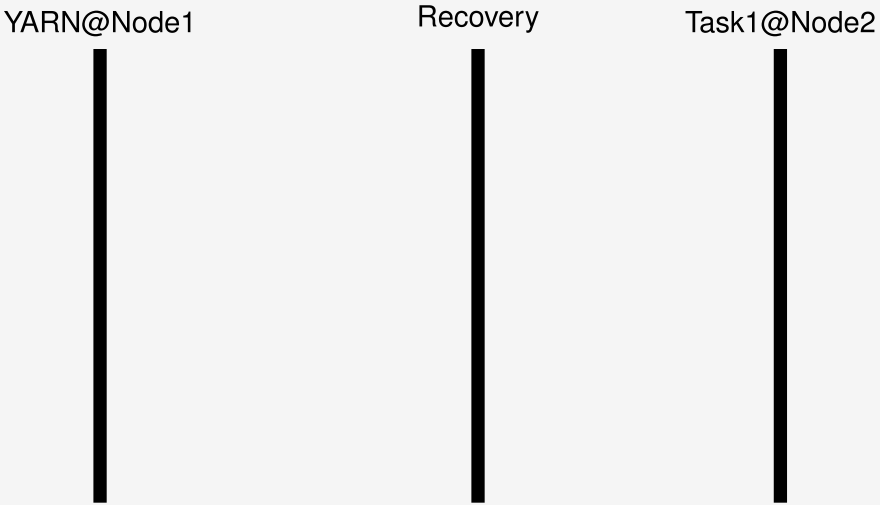


Job_1
Application_1
Task_1
Task_m
AppAttempt_1
TaskAttempt_1
TaskAttempt_m
Meta-info
Variables
Meta-info
Variables
Container0
Container1
Container_m
Node_0
Node_1
Node_N

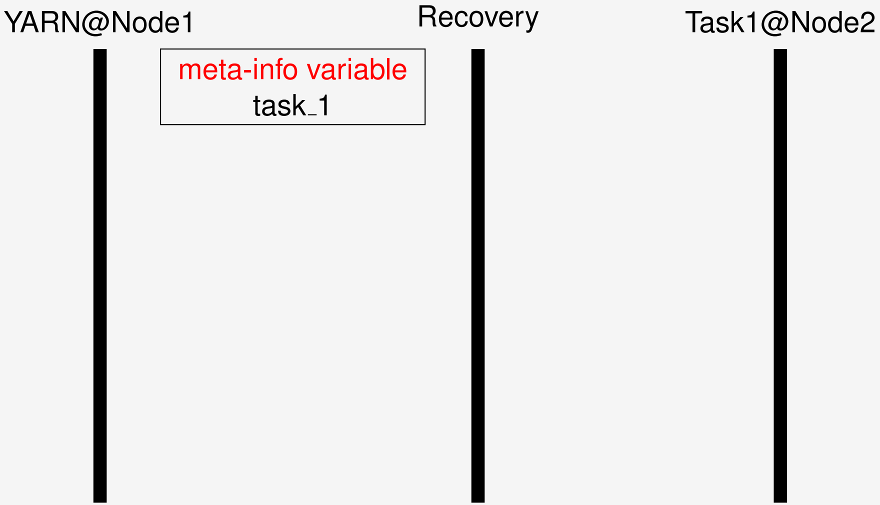
Bug Example

Node Crashes before Reading
meta-info variables

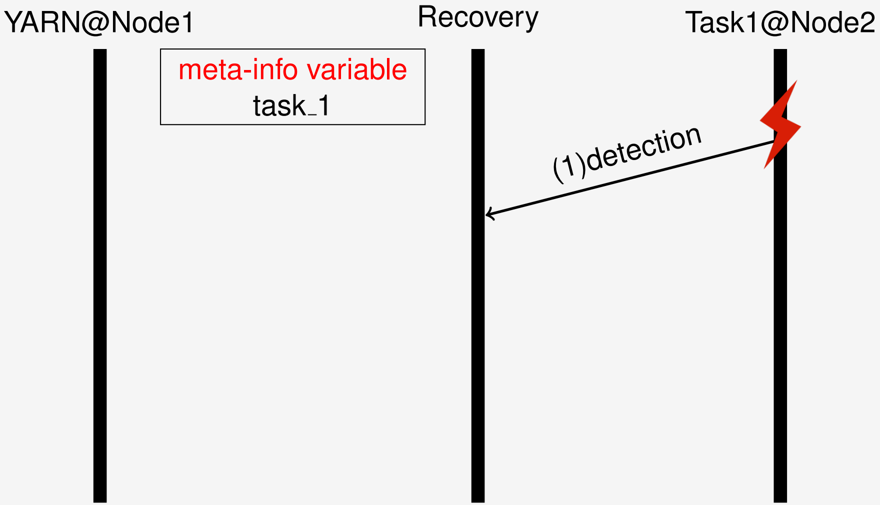
New Bug (YARN-9238) detected by CrashTuner



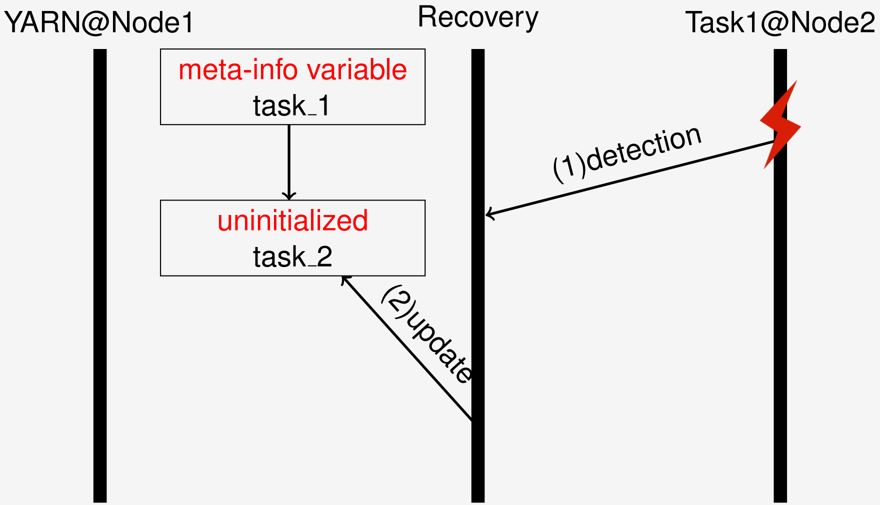
New Bug (YARN-9238) detected by CrashTuner



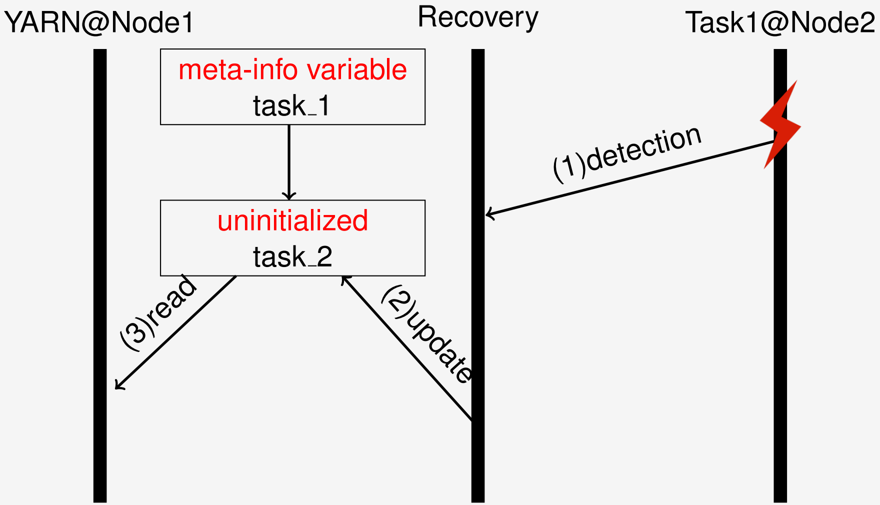
New Bug (YARN-9238) detected by CrashTuner



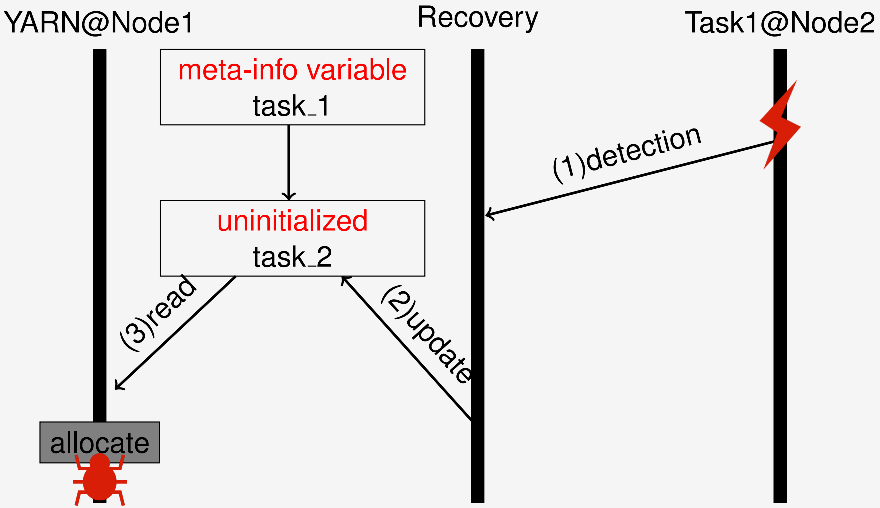
New Bug (YARN-9238) detected by CrashTuner



New Bug (YARN-9238) detected by CrashTuner

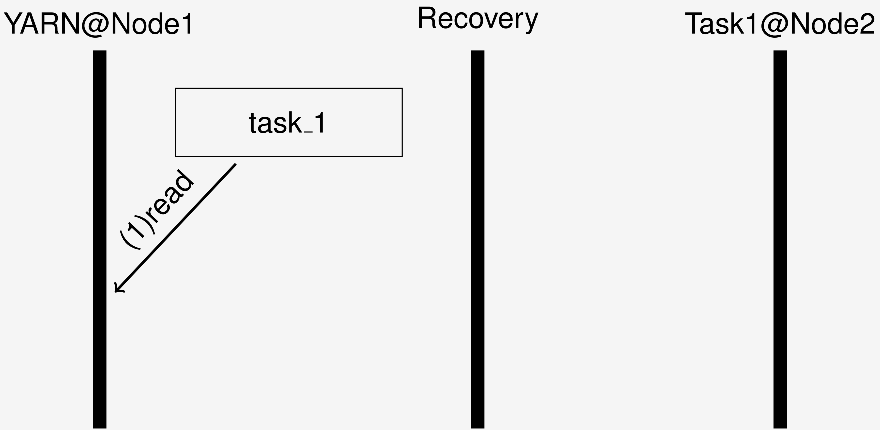


New Bug (YARN-9238) detected by CrashTuner



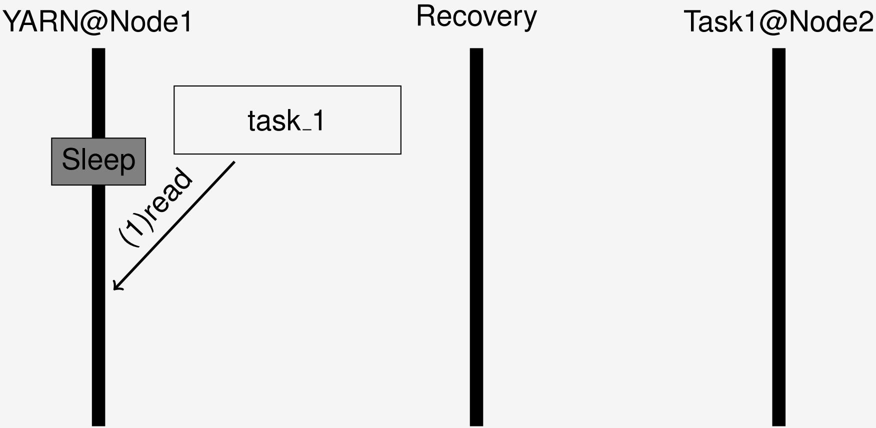
How CrashTuner Detected it?

Inject sleep and crash before reading the variable



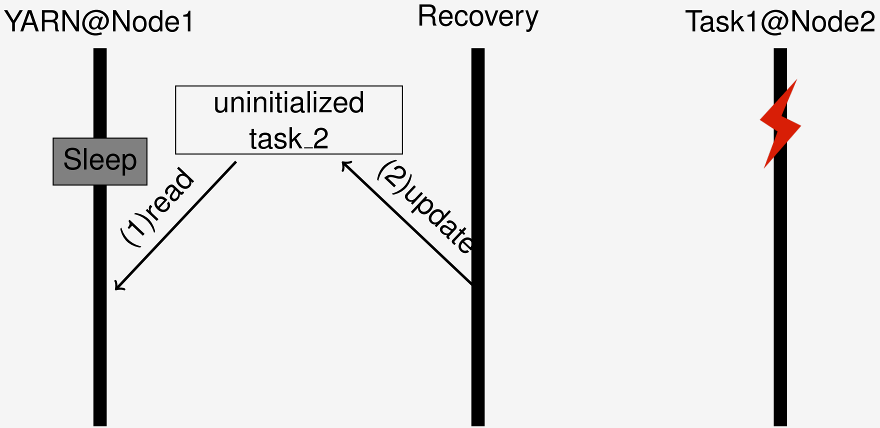
How CrashTuner Detected it?

Inject sleep and crash before reading the variable



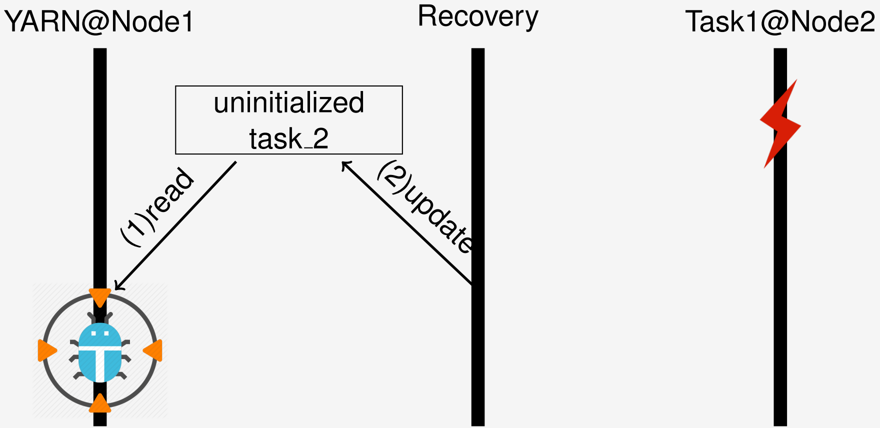
How CrashTuner Detected it?

Inject sleep and crash before reading the variable



How CrashTuner Detected it?

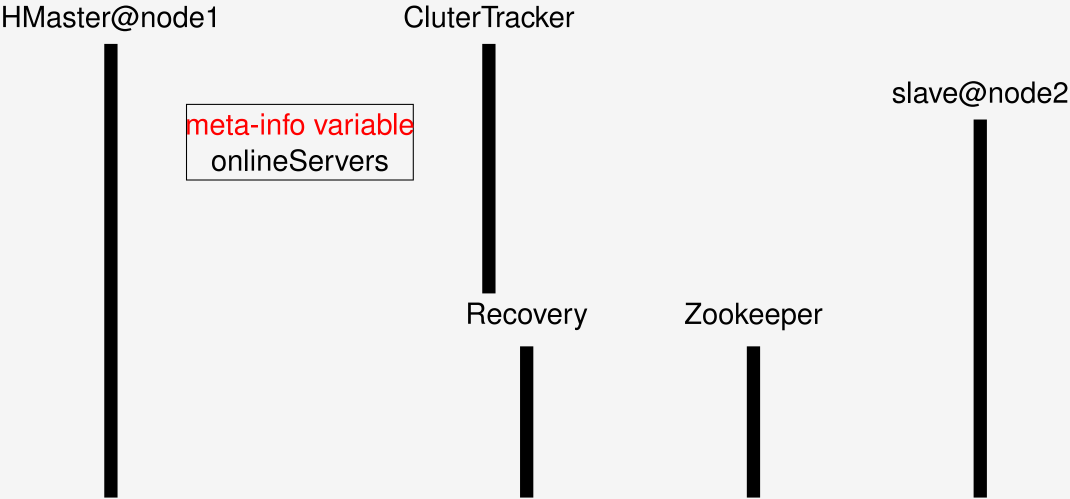
Inject sleep and crash before reading the variable



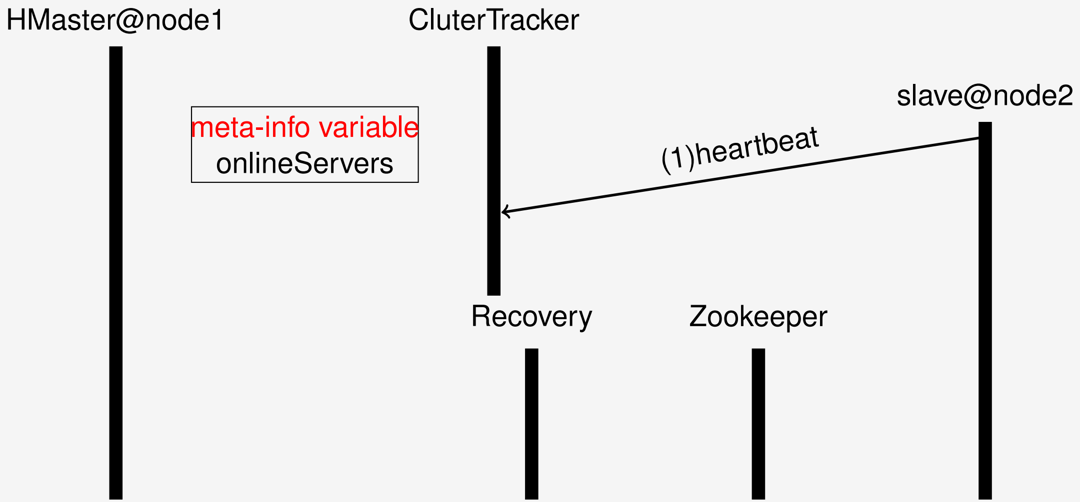
Bug Example

Node Crashes after writing meta-info variables

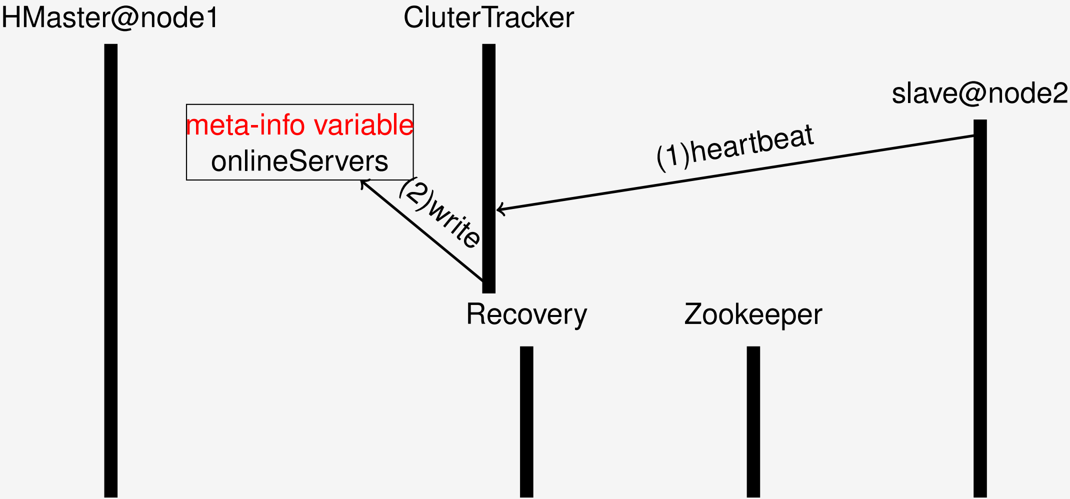
New Bug (HBASE-22041) detected by CrashTuner



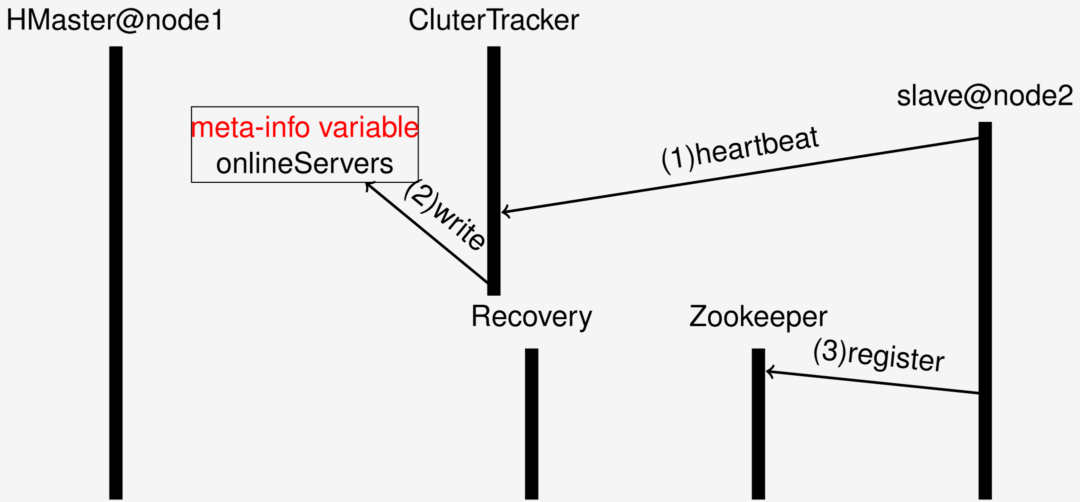
New Bug (HBASE-22041) detected by CrashTuner



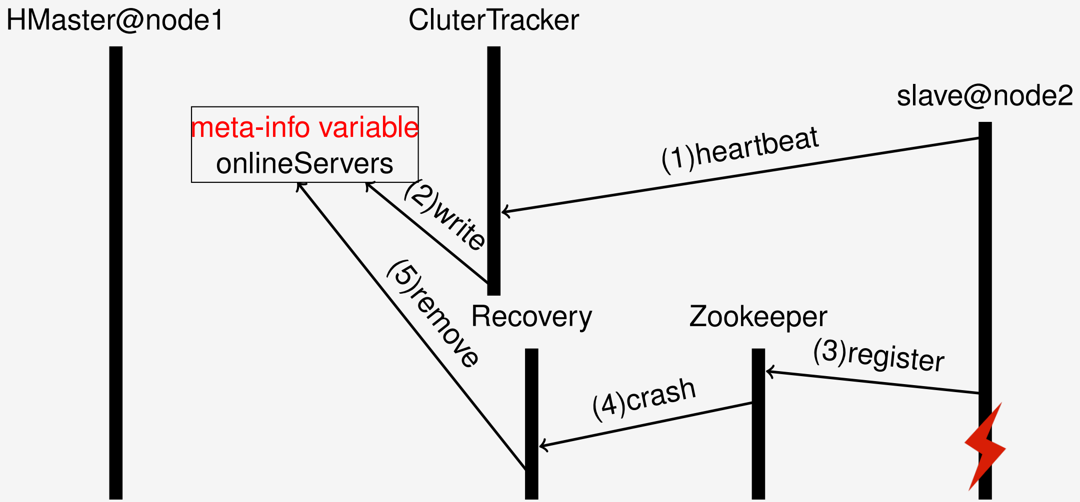
New Bug (HBASE-22041) detected by CrashTuner



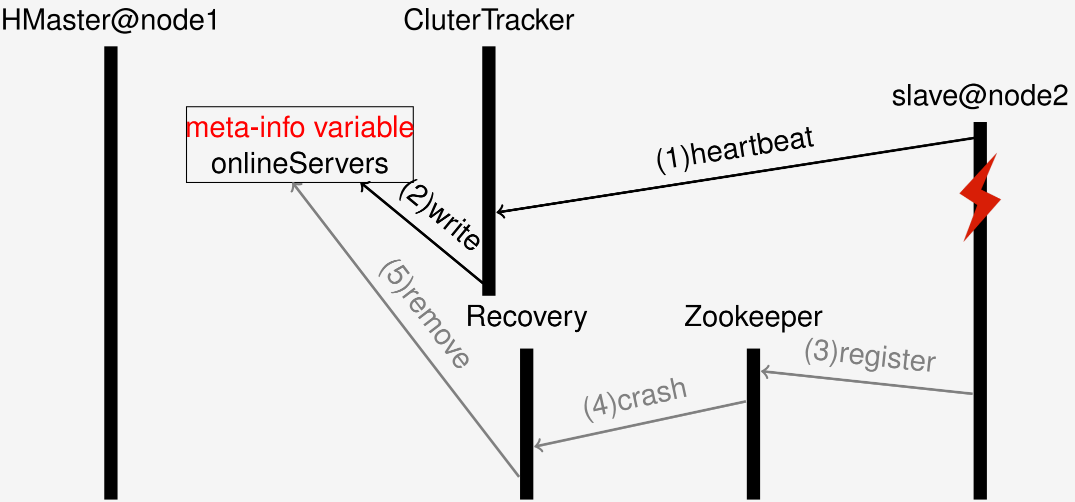
New Bug (HBASE-22041) detected by CrashTuner



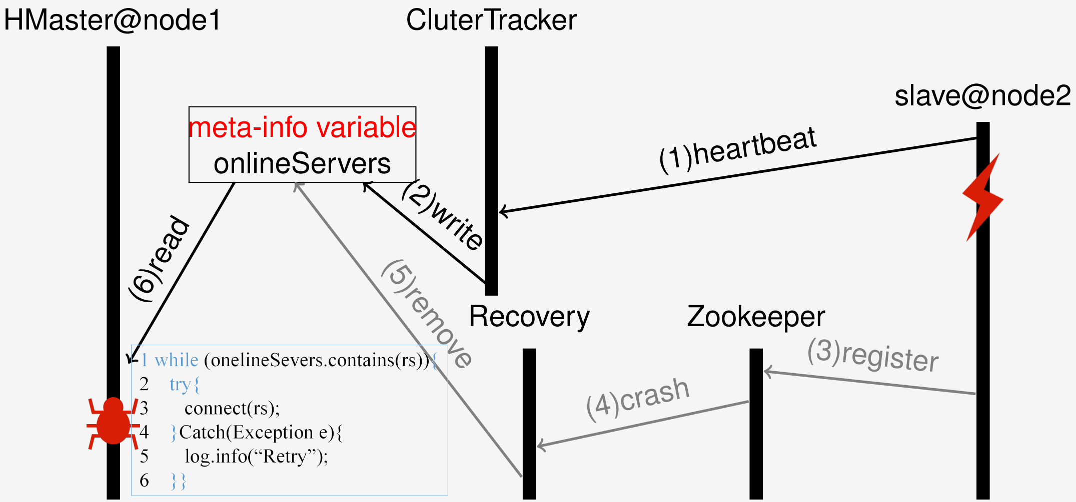
New Bug (HBASE-22041) detected by CrashTuner



New Bug (HBASE-22041) detected by CrashTuner

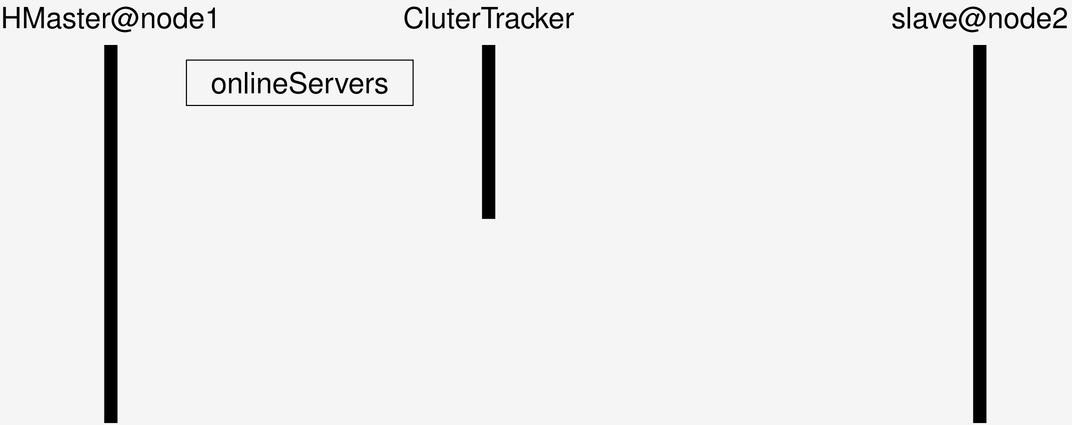


New Bug (HBASE-22041) detected by CrashTuner



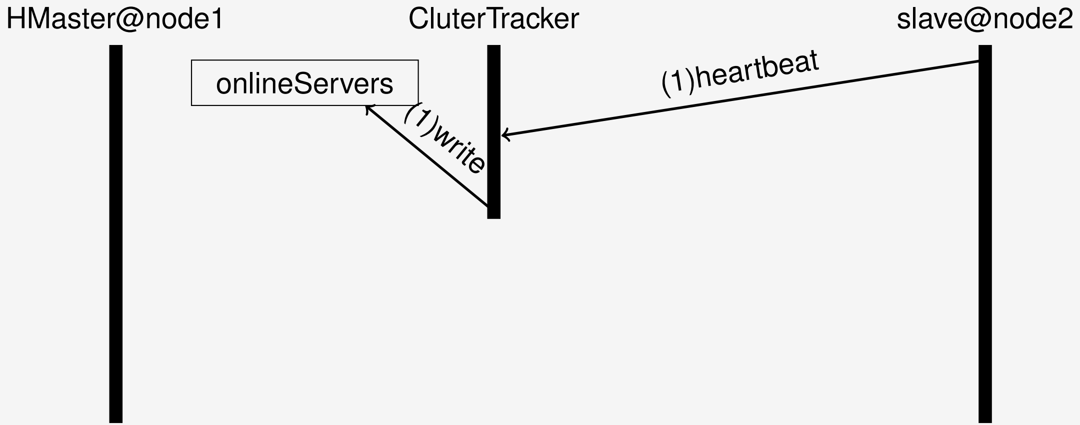
How CrashTuner detected it

Inject crash after writing the variable

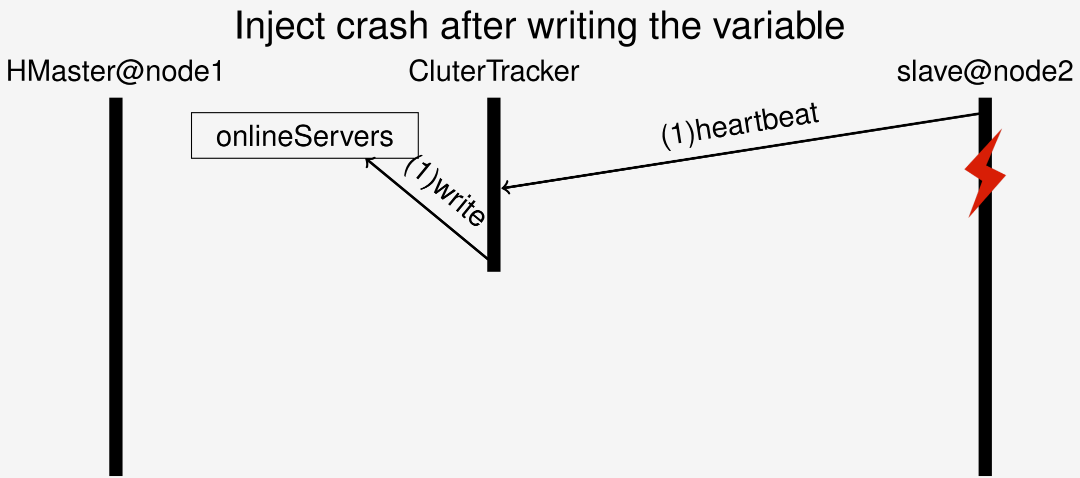


How CrashTuner detected it

Inject crash after writing the variable

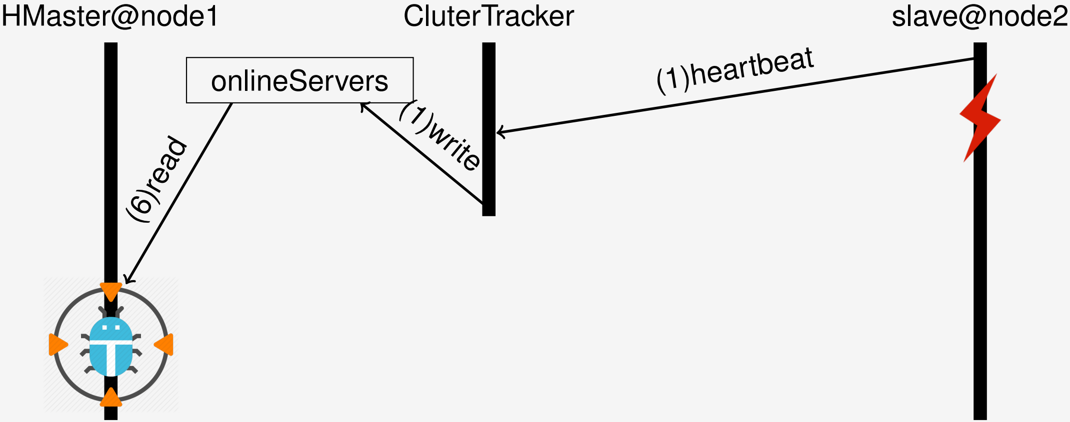


How CrashTuner detected it



How CrashTuner detected it

Inject crash after writing the variable



Meta-info variable identification

How to find meta-info variables?

Meta-info variable Identification

Node referencing variables are meta-info variables.

LOG.info("NodeManager from node " + address + " is assigned " + nodeId)

Meta-info variable Identification

Node referencing variables are meta-info variables.

LOG.info("NodeManager from node " + address + " is assigned " + nodeId)

NodeManager from node (.*) is assigned (.*)

Meta-info variable Identification

Node referencing variables are meta-info variables.

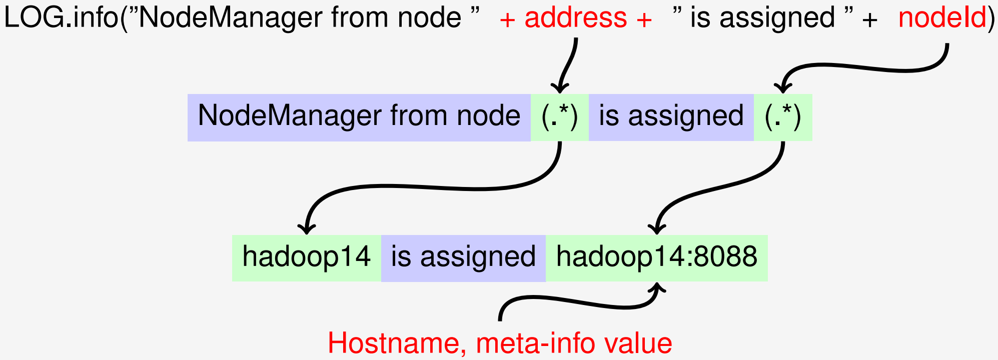
```
LOG.info("NodeManager from node " + address + " is assigned " + nodeId)
```

NodeManager from node (.*) is assigned (.*)

hadoop14 is assigned hadoop14:8088

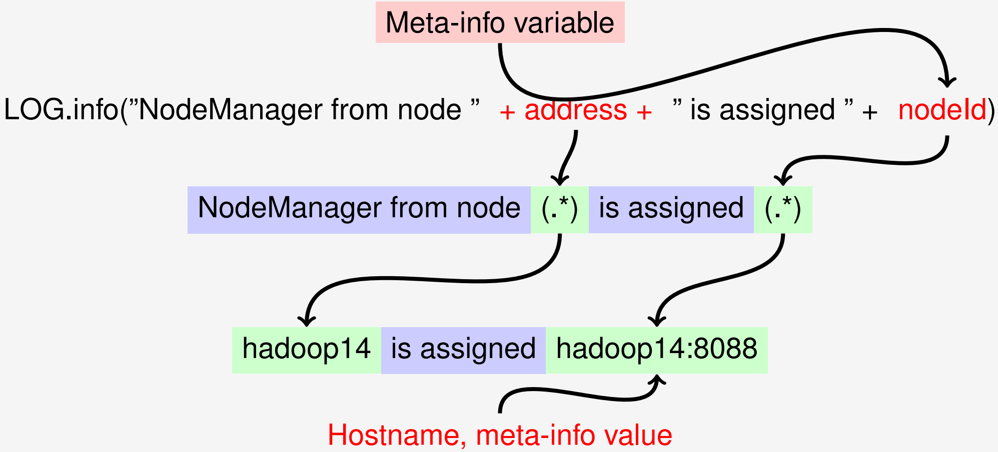
Meta-info variable Identification

Node referencing variables are meta-info variables.



Meta-info variable Identification

Node referencing variables are meta-info variables.



Meta-info variable Identification

Variables related to meta-info variable are meta-info variables.

Appearing in a same log instance.

Meta-info variable

LOG.info("Assigned Container" + containerId + " on host " + nodeId)

Meta-info variable Identification

Variables related to meta-info variable are meta-info variables.

Appearing in a same log instance.

Meta-info variable

LOG.info("Assigned Container" + containerId + " on host " + nodeId)

Meta-info variable

Meta-info variable Identification

Variables related to meta-info variable are meta-info variables.
Appearing in a same log instance.

Meta-info variable

```
LOG.info("Assigned Container" + containerId + " on host " + nodeId)
```

Meta-info variable

```
LOG.info("Assigned Container" + containerId + " to " + attempt)
```

Meta-info variable Identification

Variables related to meta-info variable are meta-info variables.
Appearing in a same log instance.

Meta-info variable

LOG.info("Assigned Container" + containerId + " on host " + nodeId)

Meta-info variable

LOG.info("Assigned Container" + containerId + " to " + attempt)

Meta-info variable

Meta-info variable Identification

- Type based static analysis to discover meta-info variables not logged.

```
/* - tracks the state of all cluster nodes*/
public class ClusterNodeTracker<N extends SchedulerNode> {
private HashMap<NodeId, N> nodes = new HashMap<>();
}
```

Meta-info type

Meta-info variable

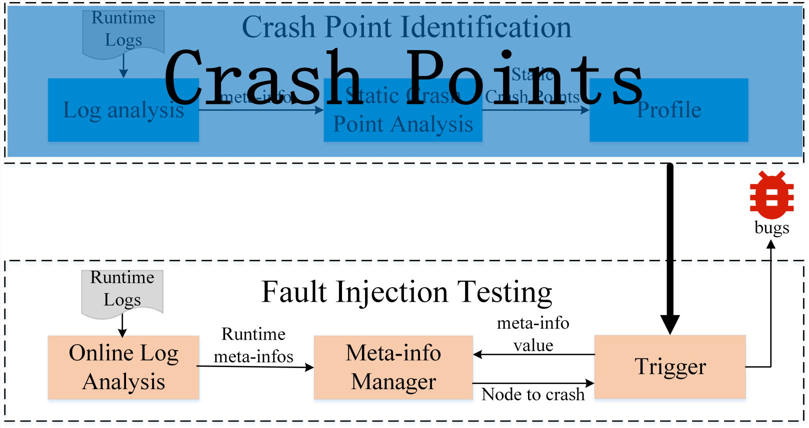
Crash Point

- Pre-read points of meta-info variables.

Crash Point

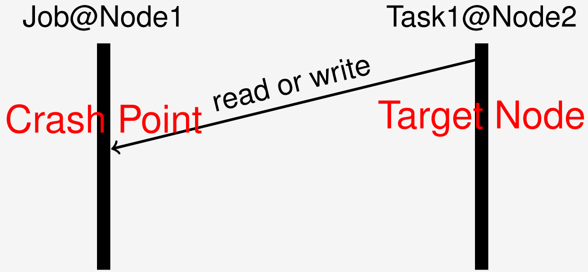
- Pre-read points of meta-info variables.
- Post-write points of meta-info variables.

Node to Crash



Node to Crash

Which node to Crash ?



Crash node2 at the crash point in node1.

Inferring the Target Node

Run time logs

```
Assigned Container_1 on hadoop14:80
Assigned Container_1 to atemmpt_1

Assigned Container_2 on hadoop15:80
Assigned Container_2 to atemmpt_2
```

→

Container_1 and attempt_1 on hadoop14

Container_2 and attempt_2 on hadoop15

Inferring the Target Node

Run time logs

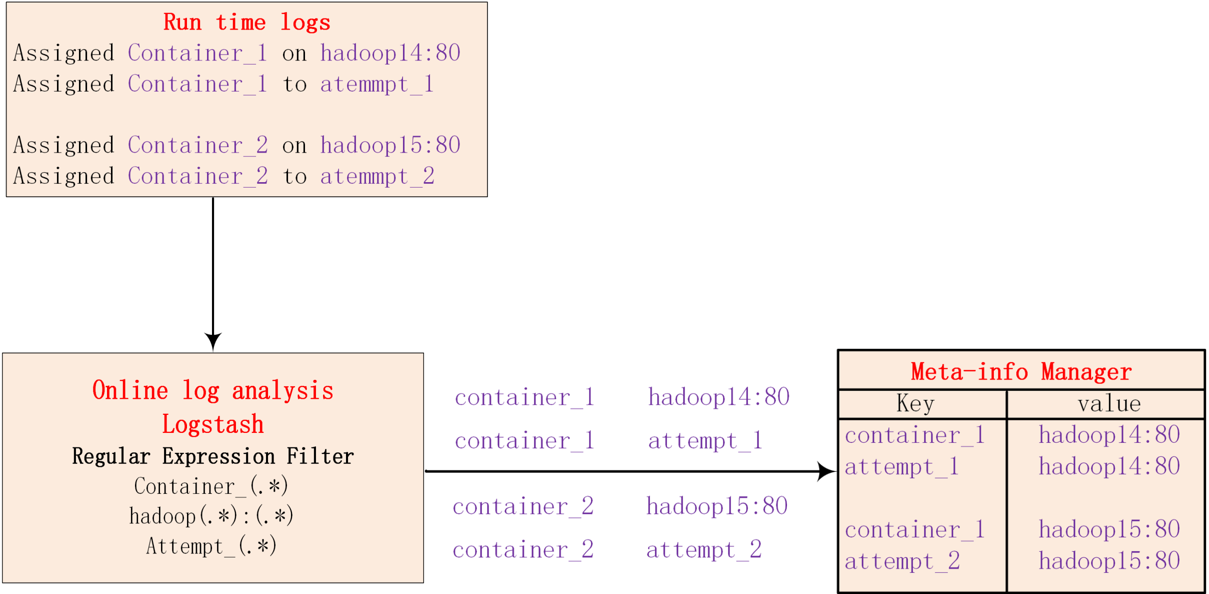
Assigned Container_1 on hadoop14:80
Assigned Container_1 to atemmpt_1

Assigned Container_2 on hadoop15:80
Assigned Container_2 to atemmpt_2

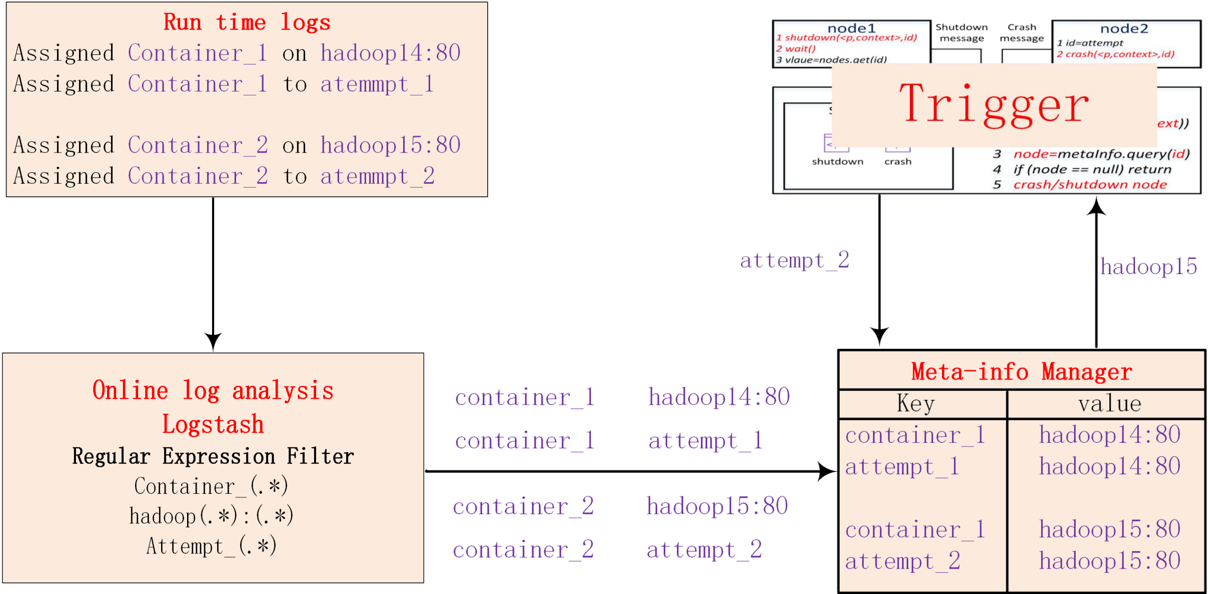
↓

Online log analysis
Logstash
Regular Expression Filter
Container_(.*)
hadoop(.*):(.*)
Attempt_(.*)

Inferring the Target Node



Inferring the Target Node



Evaluations

Table: Five distributed Systems under testing(Cassandra is not our bug-studied system).

System	Configure Change	Workload
Hadoop2/Yarn	enable opportunistic	Wordcount
HDFS	—	TestDFSIO,curl
HBase	—	PE,curl
Zookeeper	—	Smoketest
Cassandra	—	Stress

Evaluations

Table: The number of meta-info and crash point and test time.

System	# Meta-info			# Crash Points		Test time(h)
	Types	Fields	Access Points	Static	Dynamic	
Hadoop2/Yarn	107	1,251	5,109	1,524	453	17.39
HBase	34	733	4,032	920	257	8.27
HDFS	43	315	1,924	495	237	8.65
ZooKeeper	3	13	90	41	40	0.27
Cassandra	1	122	666	197	69	1.10
total	188	2,434	11,821	3,177	1,056	35.68

Evaluations

Table: The number of meta-info and crash point and test time.

System	# Meta-info			# Crash Points		Test time(h)
	Types	Fields	Access Points	Static	Dynamic	
Hadoop2/Ya[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]39
HBase	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]27
HDFS	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]5
ZooKeepe[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]27
Cassandr[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]10
total	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible].68

CrashTuner reduces 99.91% unnecessary crash points

CrashTuner reports 21 new bugs, 16 of them are already fixed

Bug ID	Type	Status	Symptom	Meta-info
YARN-1	pre-read	Fixed	Invalid event for current state of ApplicationAttempt	ContainerId
YARN-2	pre-read	Fixed	Invalid event for current state of ApplicationAttempt	ApplicationId
YARN-3	pre-read	Fixed	Scheduling the removed container	ContainerId
YARN-4	pre-read	Fixed	Allocating container to removed node	NodeID
YARN-5(2)	pre-read	Fixed	Cluster down due to using the lost node	NodeID
YARN-7(2)	pre-read	Fixed	Invalid event for current state of Container	ContainerId
YARN-9	pre-read	Fixed	Invalid event for current state of Container	ApplicationAttemptId
YARN-10	pre-read	Fixed	Resource Leak while Localizing file	ApplicationId
YARN-11	pre-read	Fixed	Allocating containers to removed ApplicationAttempt	ApplicationAttemptId
HBASE-12	post-write	Fixed	Shutdown before initialization causing abort	ServerName
HBASE-13	pre-read	Unresolved	Atomic violation causing shutdown fails	RegionInfo
HBASE-14	post-write	Unresolved	Master startup hang and print thousands of logs	ServerName
HBASE-15	post-write	Unresolved	Shutdown before initialization causing abort	ServerName
HBASE-16	pre-read	Fixed	Master Fails to become active due to LeaseException	ServerName
HDFS-17	pre-read	Fixed	Shutdown before register causing abort	DatanodeID
HDFS-18(2)	pre-read	Fixed	Request fails due to removed node	DataNodeInfo
MR-20	post-write	Unresolved	Shutdown before initialization causing abort	TaskAttemptId
CA-21	pre-read	Unresolved	Request fails due to removed node	InetAddressAndPort

Comparing to other fault injection strategies

CrashTuner report one bug in 50.29 runs within 1.70 hours.

- Random fault injection: 3 bugs, 1 bug per 5000 runs within 90.83 hours
- IO around crash injection, 1 bugs, 1 bug per 4500 runs within 156.88 hours
- All bugs can be detected by CrashTuner.

Comparing to other fault injection strategies

CrashTuner report one bug in 50.29 runs within 1.70 hours.

- Random fault injection: 3 bugs, 1 bug per 5000 runs within 90.83 hours
- IO around crash injection, 1 bugs, 1 bug per 4500 runs within 156.88 hours
- All bugs can be detected by CrashTuner.

CrashTuner is much more Efficient and Effective than random crash injection and IO around crash injection

Limitations and Future Work

- CrashTuner maybe not good enough to test system with Bad Log Quality.
  - Developer can annotate the meta-info type.
- CrashTuner only inject one crash.
  - We can extend CrashTuner to test two or more crash events.
- CrashTuner only test Java based system.
  - Our study on k8s (implemented with Golang) shows that it also have meta-info related crash-recovery bugs.
  - We are extending CrashTuner to work with System written by Golang and C++.

Relate Works

- Crash-recovery bug studies.
  - CBSDB[9],TaxDC[10], CREB[11]
- Crash-recovery bug detection
  - Fault injection:Fate[12],Fcatch[13]
  - Model checking:FlyMC[EuroSys2019],SAMC[OSDI2014]
- Log analysis for distribute systems
  - Stitch[OSDI2016], Iprof[OSDI2014]

---

[9] Haryadi S Gunawi et al. (2014). "What bugs live in the cloud? a study of 3000+ issues in cloud systems". In: *Proceedings of the ACM Symposium on Cloud Computing*. ACM, pp. 1–14.

[10] Tanakorn Leesatapornwongsa et al. (2016). "TaxDC: A Taxonomy of Non-Deterministic Concurrency Bugs in Datacenter Distributed Systems". In: *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '16. Atlanta, Georgia, USA: ACM, pp. 517–530. ISBN: 978-1-4503-4091-5. DOI: 10.1145/2872362.2872374. URL: http://doi.acm.org/10.1145/2872362.2872374.

[11] Yu Gao et al. (2018). "An Empirical Study on Crash Recovery Bugs in Large-Scale Distributed Systems". In: *Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2018.

[12] Haryadi S Gunawi et al. (2011). "FATE and DESTINI: A framework for cloud recovery testing". In: *Proceedings of NSDI'11: 8th USENIX Symposium on Networked Systems Design and Implementation*, p. 239.

[13] Haopeng Liu et al. (2018). "Fcatch: Automatically detecting time-of-fault bugs in cloud systems". In: *ACM SIGPLAN Notices* 53.2, pp. 419–431.

Conclusion

Abstraction is so fundamental that sometimes we forget its importance![14]
—Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau

[14] Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau (2018). *Operating Systems: Three Easy Pieces*. 1.00. Arpaci-Dusseau Books.

Conclusion

Abstraction is so fundamental that sometimes we forget its importance![14]

—Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau

***Meta-info* is a well-suited abstraction for distributed systems!**

[14] Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau (2018). *Operating Systems: Three Easy Pieces*. 1.00. Arpaci-Dusseau Books.

Thank you!

Any Questions?

backup slides

How to find the meta-info value at crash point

```
RPC.send(masterContainer.getId());
```

Crash Point

```
private void launch() throws IOException, YarnException {
    ContainerId masterContainerID = masterContainer.getId();

    ApplicationSubmissionContext applicationContext =
      application.getSubmissionContext();

    LOG.info("Setting up container " + masterContainer
        + " for AM " + application.getAppAttemptId());
}
```

```
public ContainerId getId() {
  return this.containerId;
}
```

How to find the meta-info value at crash point

```java
public ContainerId getId() {
  ContainerProtoOrBuilder p = viaProto ? proto : builder;
  if (this.containerId != null) {
    return this.containerId;
  }
  if (!p.hasId()) {
    return null;
  }
  this.containerId = convertFromProtoFormat(p.getId());
  return this.containerId;
}
```

Side effect?

How to find the meta-info value at crash point

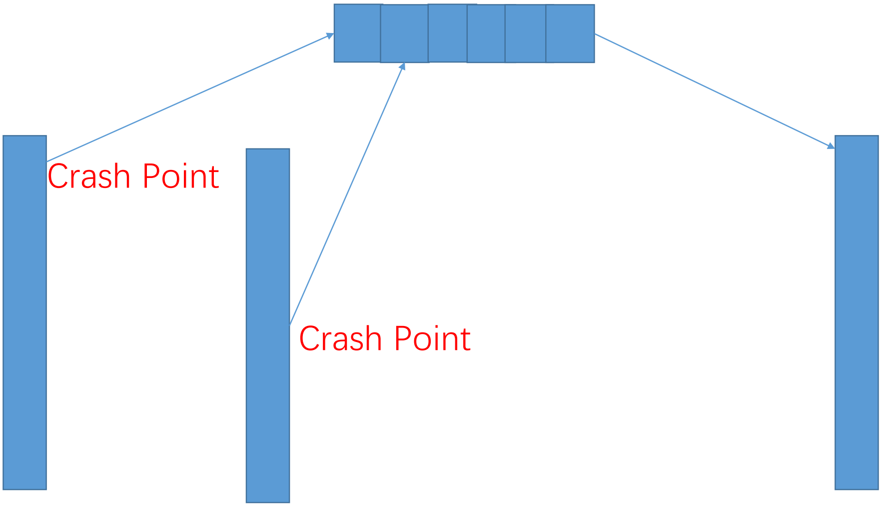
```
RPC.send(masterContainer);
```

```
private void launch() throws IOException, YarnException {
    ContainerId masterContainerID = masterContainer.getId();

    ApplicationSubmissionContext applicationContext =
      application.getSubmissionContext();

    LOG.info("Setting up container " + masterContainer
        + " for AM " + application.getAppAttemptId());
}
```

Avoid lock

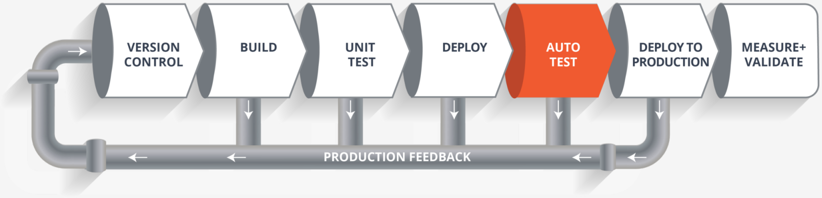
```
public void setNodeId(NodeId nodeId)
     lock()
     this.nodeId = nodeId;
     unlock()
}

public NodeId getNodeId( ) {
     lock()
     return nodeId;
     unlock()
}

public void lunch(){
     //Crash Point
     NodeId nodeId = getNodeId();
     node.getHttpAddress();
}
```

For Event Handler



CI/CD

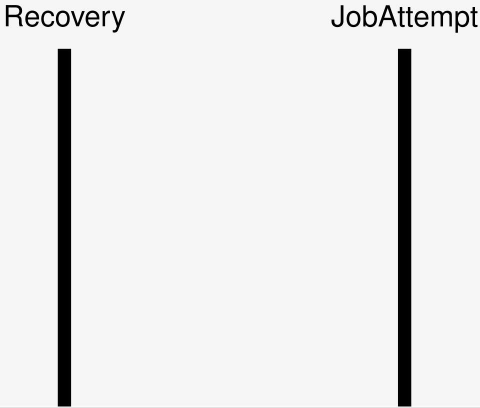
CrashTuner can be easily embedded in CI/CD Pipeline at Auto Test phase.



Discussion

Why Random fault injection is less Efficient and Effective?

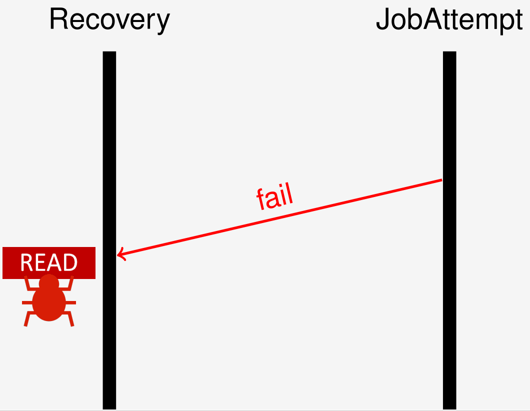
Random approach is less time sensitive.
New bug: YARN-9238



Discussion

Why Random fault injection is less Efficient and Effective?

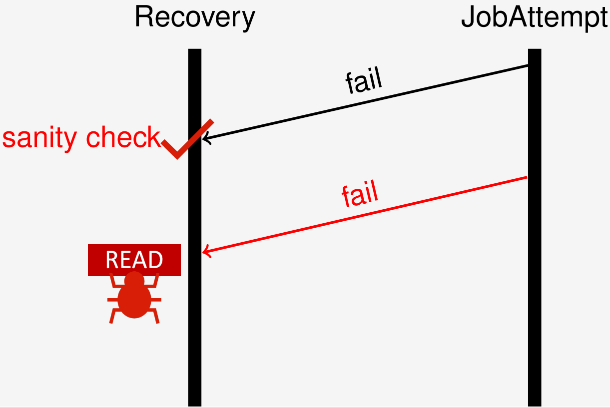
Random approach is less time sensitive.
New bug: YARN-9238



Discussion

Why Random fault injection is less Efficient and Effective?

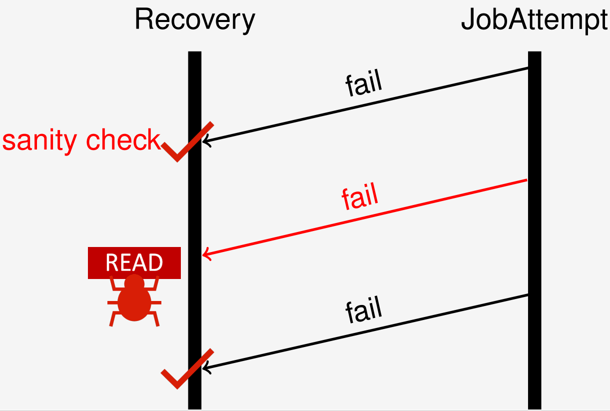
Random approach is less time sensitive.
New bug: YARN-9238



Discussion

Why Random fault injection is less Efficient and Effective?

Random approach is less time sensitive.
New bug: YARN-9238



Discussion

Why Random fault injection is less Efficient and Effective?

Random approach is less time sensitive.
New bug: YARN-9238

Recovery JobAttempt

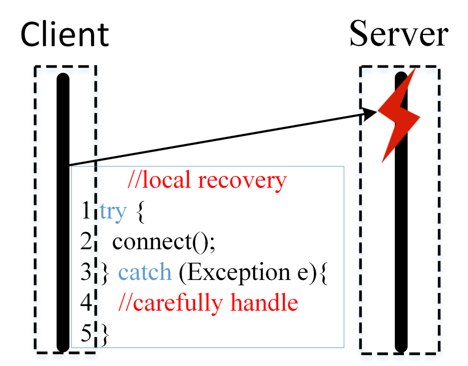
Discussion

Why IO around Fault injection is less Efficient and Effective?

Discussion

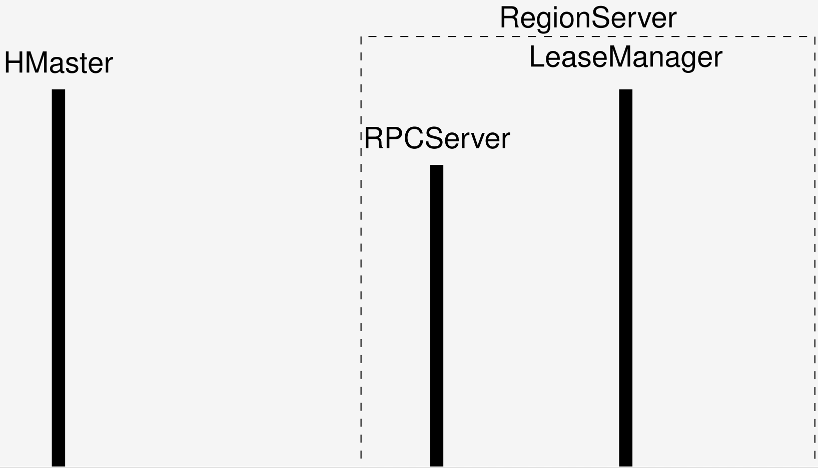
Why IO around Fault injection is less Efficient and Effective?

- IO Crash Points are far away the real crash point.
- Local recovery: Developers instinctively handle the error related IO operation.



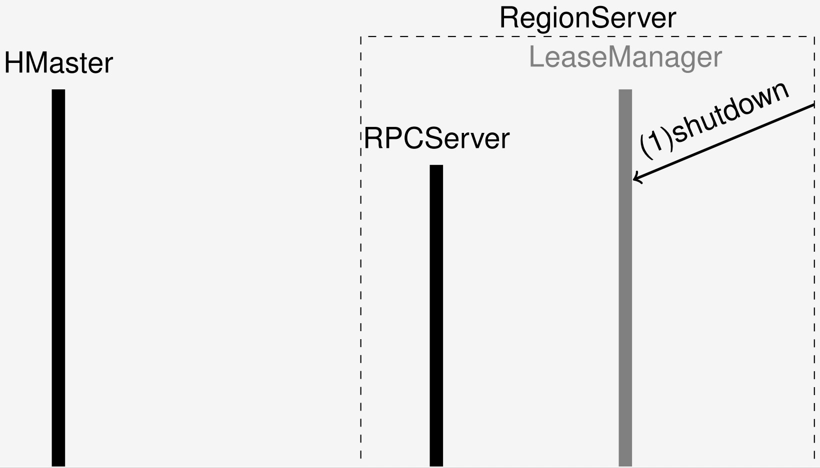
Bug example of shutdown:HBASE-22017

Shutdown can simulate software failure[Socc2014] and gray failure[OSDI2018, ATC2019]: one component is sick,left are health.



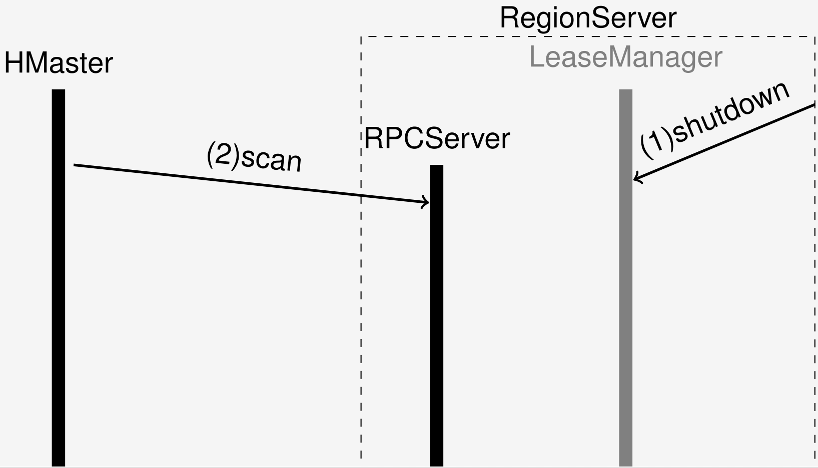
Bug example of shutdown:HBASE-22017

Shutdown can simulate software failure[Socc2014] and gray failure[OSDI2018, ATC2019]: one component is sick,left are health.



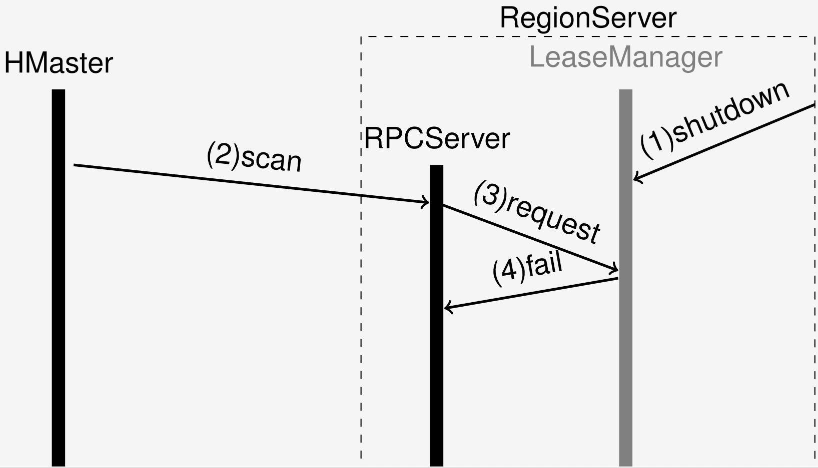
Bug example of shutdown:HBASE-22017

Shutdown can simulate software failure[Socc2014] and gray failure[OSDI2018, ATC2019]: one component is sick,left are health.



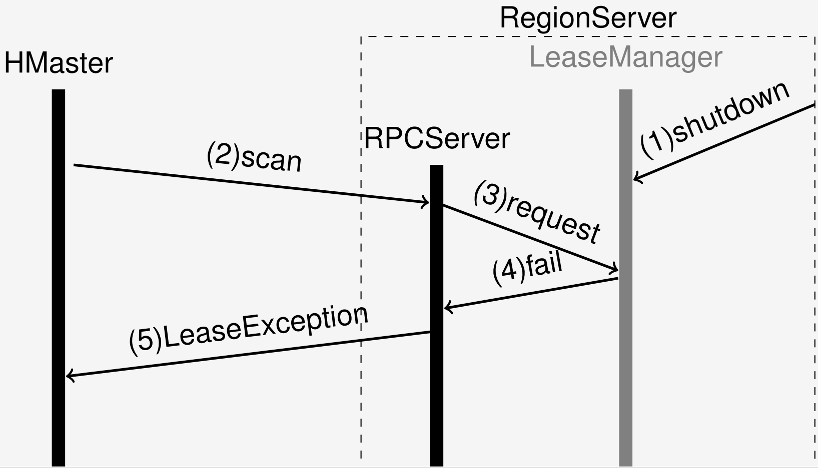
Bug example of shutdown:HBASE-22017

Shutdown can simulate software failure[Socc2014] and gray failure[OSDI2018, ATC2019]: one component is sick,left are health.



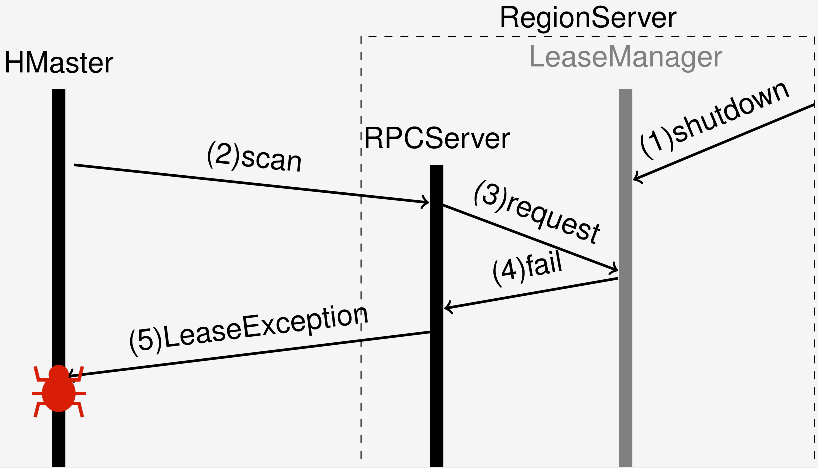
Bug example of shutdown:HBASE-22017

Shutdown can simulate software failure[Socc2014] and gray failure[OSDI2018, ATC2019]: one component is sick,left are health.



Bug example of shutdown:HBASE-22017

Shutdown can simulate software failure[Socc2014] and gray failure[OSDI2018, ATC2019]: one component is sick,left are health.



Q1: Profile:Dynamic Crash Point

- Filtering un-executed Static Crash Points.
- Calling context of One Crash Point.

Crash Point = {CLASS, Method, LineNumber,PreOrPost, Call Statck}