PRECISELY TRACE A REQUEST IN CLOUD ENVIRONMENT

POLYTECHNIQUE MONTRÉAL
LE GÉNIE EN PREMIÈRE CLASSE

Progress report meeting
December 2013

Phuong Tran Gia
gia-phuong.tran@polymtl.ca

Under the supervision of Prof. Michel R. Dagenais
Dorsal Laboratory, Polytechnique Montréal

Outline

- Research objectives
- Literature review
- Methodology
- Future work
- Challenges

Research objectives

Research objectives

- Research questions
- Possible contributions

Research questions

- “How to trace and analyze a request in cloud environment at the kernel level? If impossible at the kernel level, how to minimally instrument the application’s source code and middleware?”

- “Is it possible to predict the associated overhead of tracing tools based on the traffic workloads?”

Possible contributions

- The overall objective:

  “Provide efficient analysis algorithms and tools in order to know exactly how a request is serviced in modern cloud-based systems and to collect the time spent on many services in the cloud”.

- Possible contributions:
  - An automated method of following a request without instrumenting the application’s source code or special RPC libraries itself in cloud environment.
  - A method to select and study user requests of interest based on the system model and other parameter, such as: set of activated probes, sampling rate or traffic workload.
  - Algorithms and views to analyze the Asynchronous RPC transaction data in cloud environment

Literature review

Literature review

- **Approaches to white-box tracing**
  - require the knowledge of application or middleware
  - Tracepoints
  - Request ID propagation
- **Statistical approaches to black-box tracing**
  - Events are collected from communications: incoming, outgoing messages.
  - Imprecise method.
- **Precise approaches to black-box tracing**
  - Does not require to modify the application's source code
  - Insufficient to obtain the traditional trace data.
  - 3 solutions so far: vPath[4], BorderPatrol[5], PreciseTracer[6]

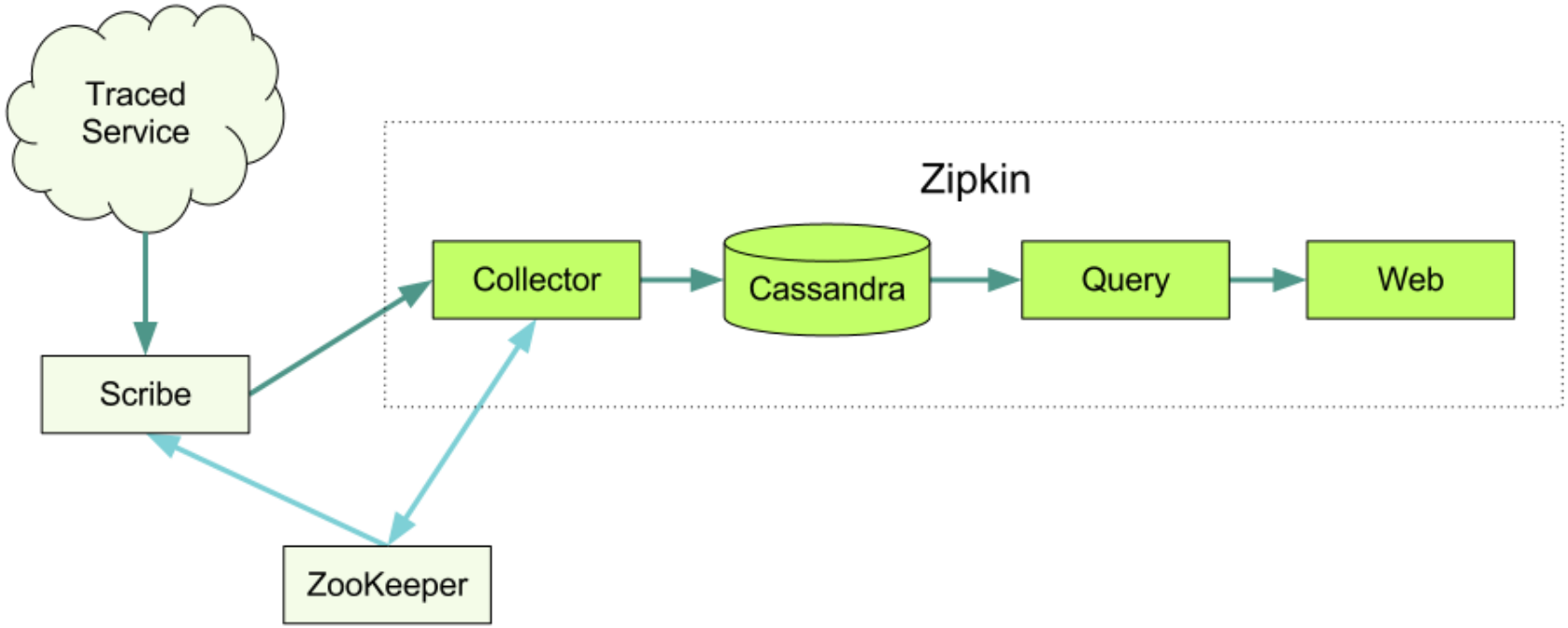
Literature review

Tracing Infrastructures	Use cases	Accuracy Tracing	Sampling technique	Scalability	Operation mode
Dapper	Distributed profiling Diagnosing bottlenecks	Precise	Yes (Adaptive sampling)	Yes	Offline Online(limited)
Zipkin	Distributed profiling Diagnosing bottlenecks	Precise	Yes (Uniform sampling)	Yes	Offline
Magpie	Anomaly detection Workload modelling	Precise	No	No	Online/Offline
X-Trace	Diagnosing bottlenecks	Precise	No		
Whodunit	Distributed profiling	Precise	No		
Pip	Diagnosing bottlenecks	Precise	No	No	Offline
Pinpoint	Anomaly detection Diagnosing bottlenecks	Precise	No		Online
Stardust	Diagnosing bottlenecks Workload modelling	Precise	No		
E2Eprof	Diagnosing bottlenecks Anomaly detection	Imprecise	No	No	Online
Project 5	Diagnosing bottlenecks	Imprecise	No	No	Offline
WAP5	Diagnosing bottlenecks	Imprecise But per process	No	No	Offline
BorderPatrol	Diagnosing bottlenecks	Precise	No	No	Offline
vPath	Diagnosing bottlenecks	Precise	No	No	Offline
PreciseTracer	Diagnosing bottlenecks	Precise	Yes	Yes	Online

Zipkin[2] – A distributed tracing framework

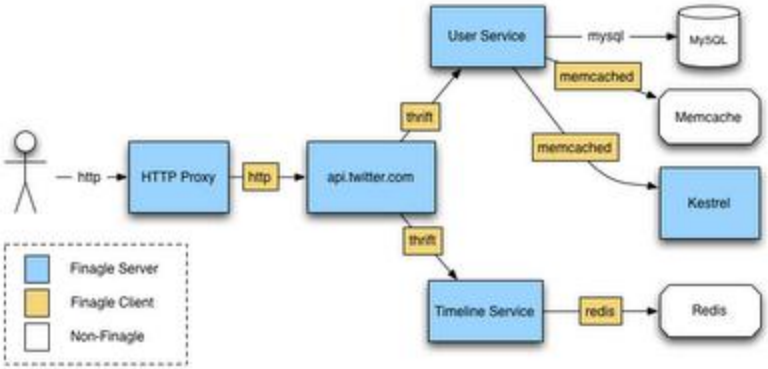
- Why Zipkin?
  - Helping developers gain deeper knowledge about how certain requests perform in a distributed system.
  - Helping developers gather timing data for the disparate services at Twitter.

Zipkin – Architecture



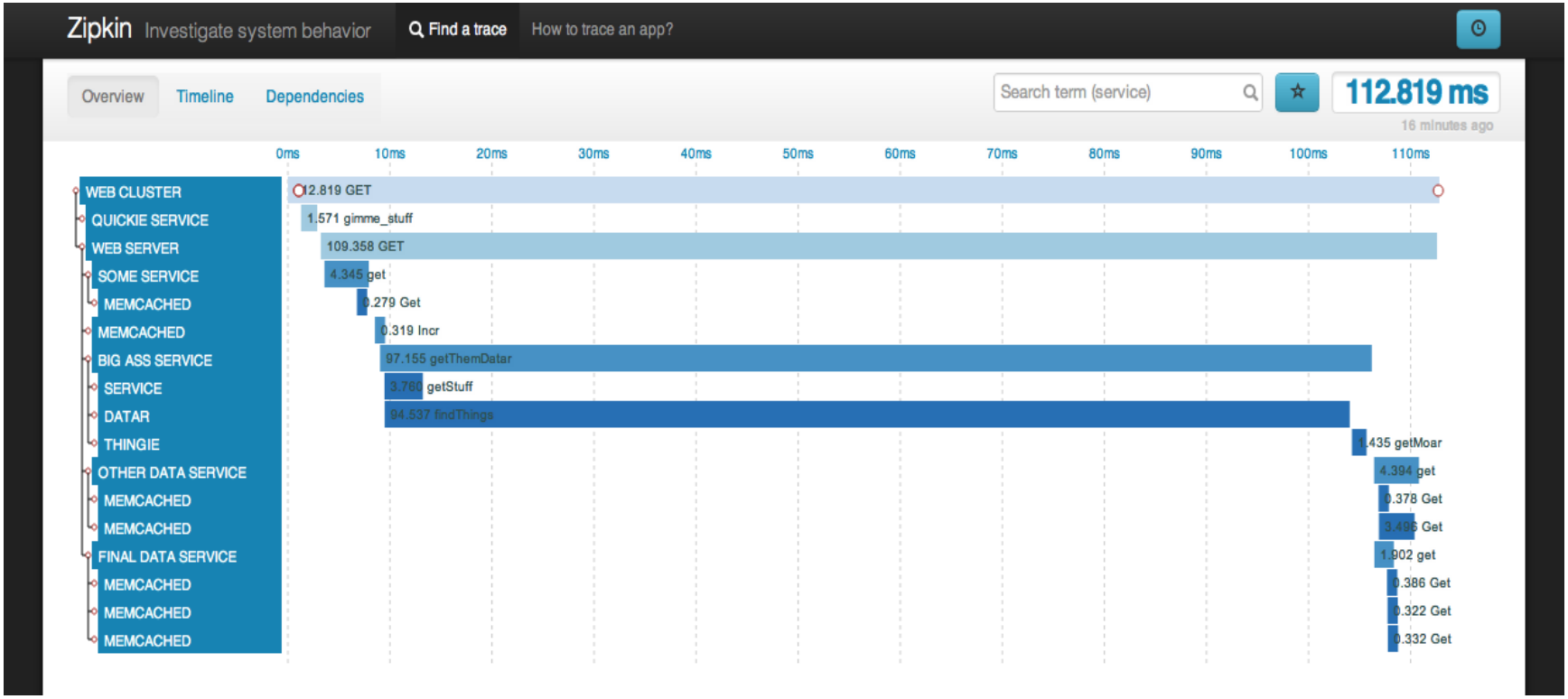
Finagle in Zipkin

- A core module of Zipkin.
- Finagle is an asynchronous network stack for the JVM that we can use to build asynchronous Remote Procedure Call (RPC) clients and servers in Java, Scala, or any JVM-hosted language.



github.com/twitter/finagle

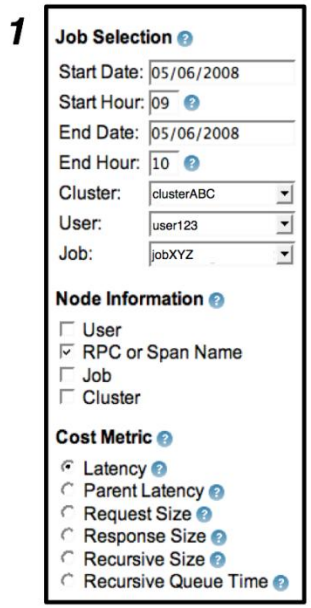
Zipkin UI

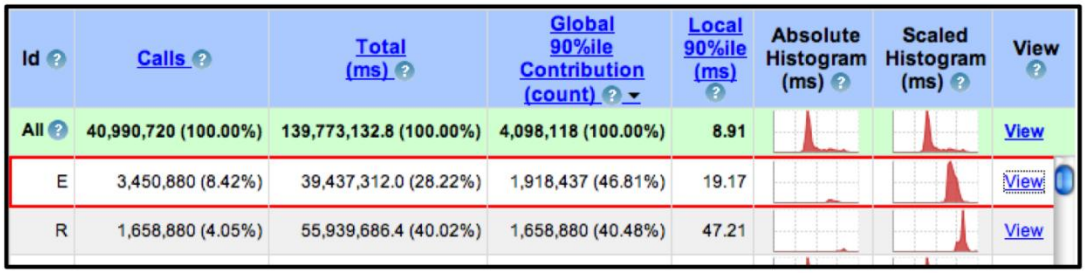

Zipkin
Investigate system behavior
Find a trace
How to trace an app?
Overview
Timeline
Dependencies
Search term (service)
112.819 ms
16 minutes ago
0ms
10ms
20ms
30ms
40ms
50ms
60ms
70ms
80ms
90ms
100ms
110ms
WEB CLUSTER
QUICKIE SERVICE
WEB SERVER
SOME SERVICE
MEMCACHED
MEMCACHED
BIG ASS SERVICE
SERVICE
DATAR
THINGIE
OTHER DATA SERVICE
MEMCACHED
MEMCACHED
FINAL DATA SERVICE
MEMCACHED
MEMCACHED
MEMCACHED
12.819 GET
1.571 gimme_stuff
109.358 GET
4.345 get
0.279 Get
0.319 Incr
97.155 getThemDatar
3.760 getStuff
94.537 findThings
1.435 getMoar
4.394 get
0.378 Get
3.495 Get
1.902 get
0.386 Get
0.322 Get
0.332 Get

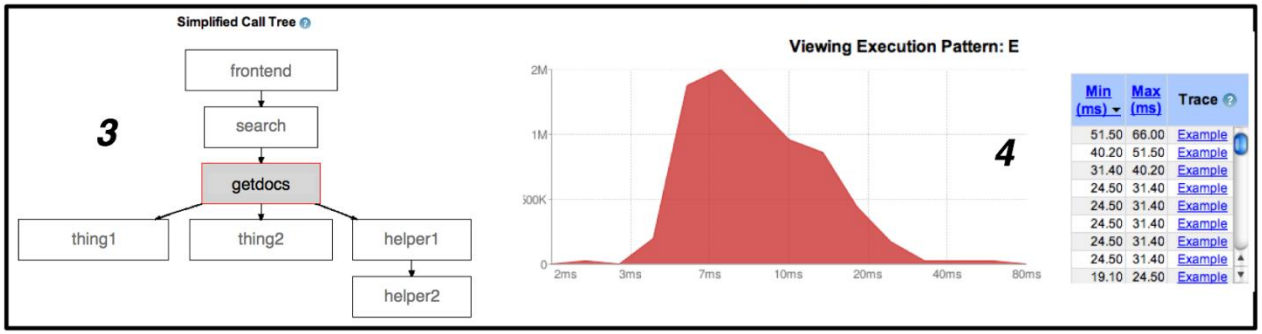
Google's Dapper

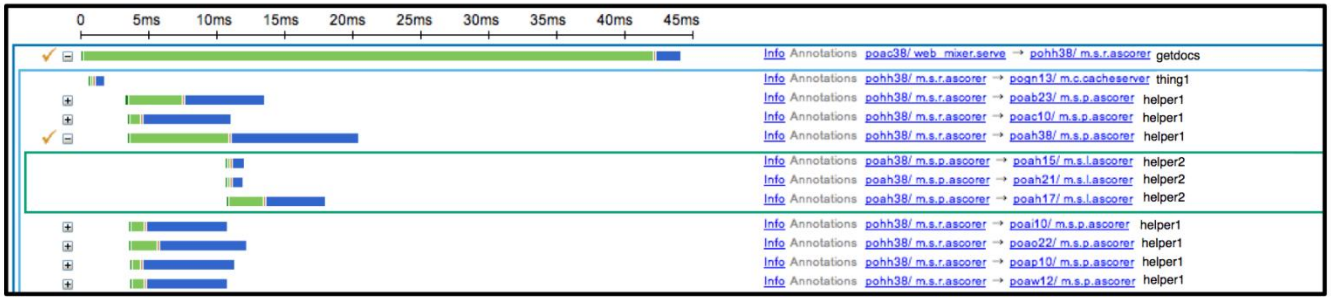
Google's Dapper[1]

- Collect traces from production requests
- Low overhead
- Minimum of extra work for developer

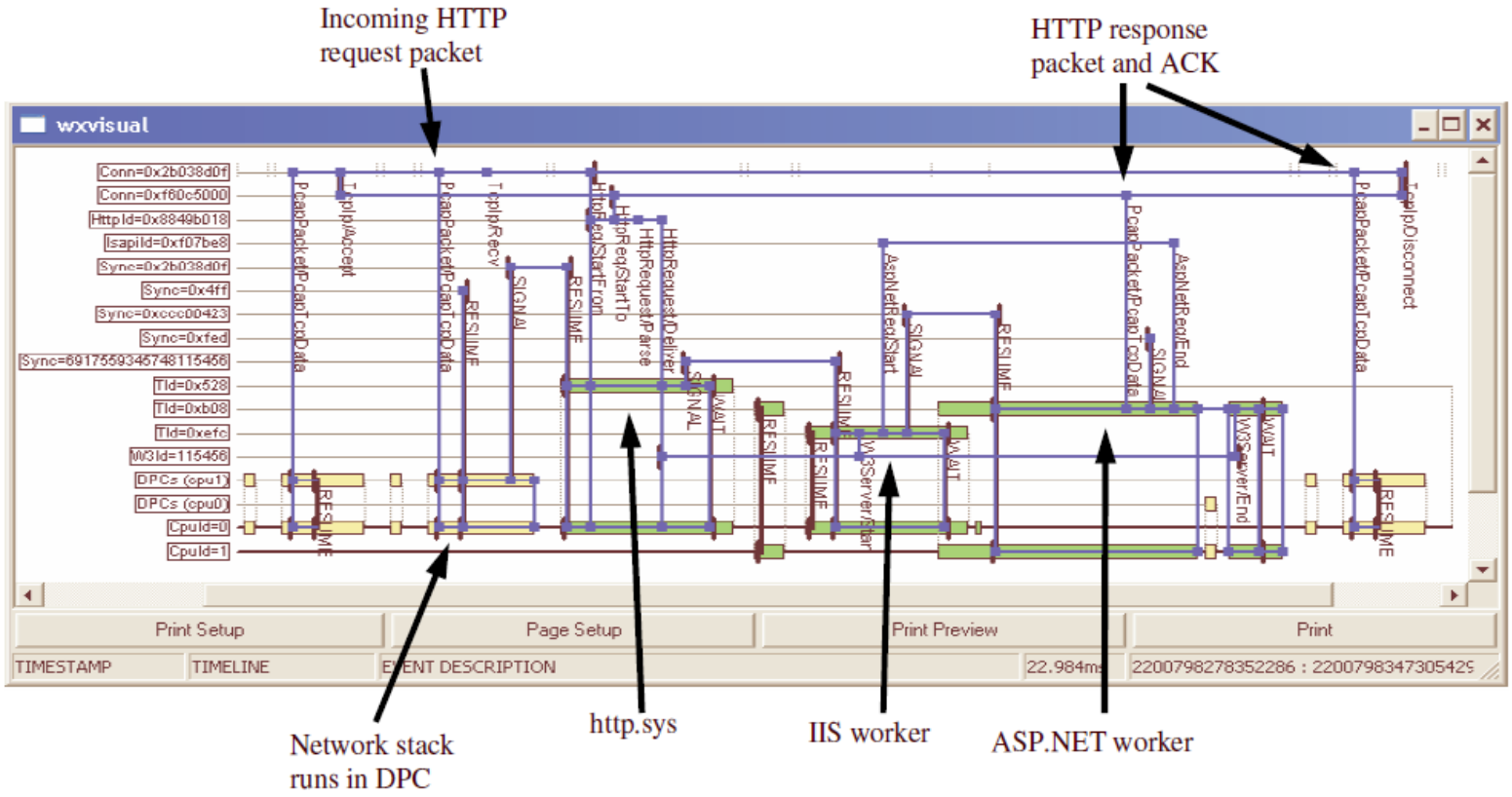








Magpie[3]


Incoming HTTP request packet
HTTP response packet and ACK
wxvisual
Conn=0x2b038d0f
Conn=0xf60c5000
HttpId=0x8849b018
IsapiId=0xf07be8
Sync=0x2b038d0f
Sync=0x4ff
Sync=0xccc00423
Sync=0xfed
Sync=69175593457481154 56
Tid=0x528
Tid=0xb08
Tid=0xefc
W3Id=115456
DPCs (cpu1)
DPCs (cpu0)
CpuId=0
CpuId=1
Print Setup
Page Setup
Print Preview
Print
TIMESTAMP
TIMELINE
EVENT DESCRIPTION
22.984ms
Network stack runs in DPC
http.sys
IIS worker
ASP.NET worker

Methodology

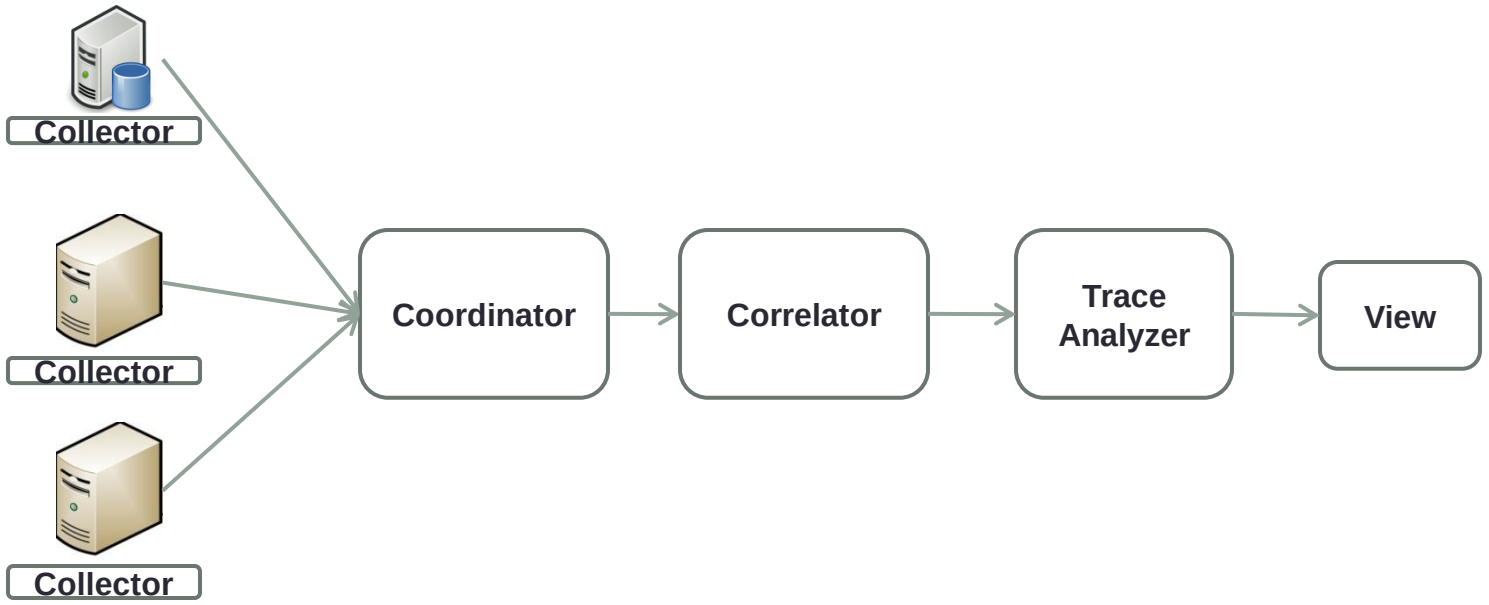
Methodology

- System model and problem statement
- System architecture
- Temporal Join Algorithm
- Case study

System model and problem statement

- System model:
  - Large clusters: disparate services, each service is running on one node.
  - Each node (component) can serve many requests in a certain period.
  - Treat each component as a grey-box (minimize the instrumentation of application source code and middleware).
- Problem statement:
  - Asynchronous communication between nodes (components).
  - Support event-driven architecture (like Finagle at Twitter)
  - On each component:
    - More than one threads to serve a request.
    - threads can be used many times (reused).

System architecture



System architecture

- Collector:
  - What information do we capture?
    - Inter-process communication between components:
      - *Domain name, Timestamp, Sender_IP, Sender_Port, Receiver_IP, Receiver_Port, Message Size*
    - Track the thread **(Tid)** performing the system call over the incoming, outgoing TCP connections.
    - Resource accounting for above thread (**Tid**) such as context switching and blocking on network.

Approach

- How do we trace a request?
  - Design goals:
    - Determine which events from the stream pertain to a specific request?
    - Minimize the instrumentation for several types of RPC.
  - Problems:
    - The thread may post the same event for any number of different requests.
    - More than one threads to serve a request.
  - Solution: Using event-schema to describe event relationships for the particular application service, middleware of interest. (do not require to instrument the application or middleware).

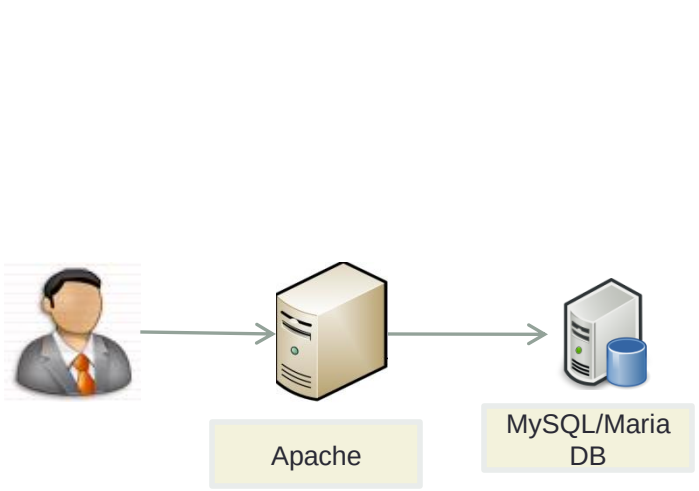
Algorithm

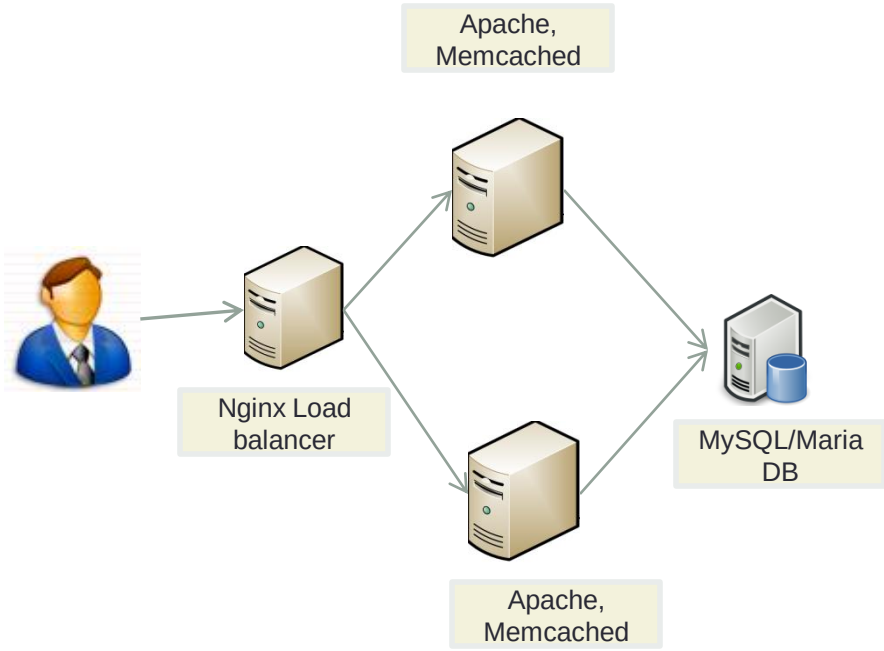
- Event-schema:
  - Specify which attribute of event connect to other events.
  - Identify each execution interval: BIND_START, BIND_STOP, BIND_BASIC[3]
  - Example:
    - EVENT(“HttpRequest", "Start");
    - ATTRIBUTE("TId", BIND_START, 0);
    - EVENT(“HttpRequest", "Start");
    - ATTRIBUTE("TId", BIND_BASIC, 0);
    - ATTRIBUTE(" **WebserverID** ", BIND_BASIC, 0);
    - EVENT(" HttpRequest ", "End");
    - ATTRIBUTE("ThreadId", BIND_BASIC, 0);
    - ATTRIBUTE(“**WebserverID**", BIND_BASIC, 0);
    - EVENT(“HttpRequest", "Start");
    - ATTRIBUTE("TId", BIND_STOP, 0);

- Temporal join:
  - Following of a request as queries against a temporal database: each table holds the events of a given type.
  - During a valid-interval (BIND_START and BIND_STOP) , events are joined together based on the join attribute between them (**WebserverID**) in above example of event schema.

Case study

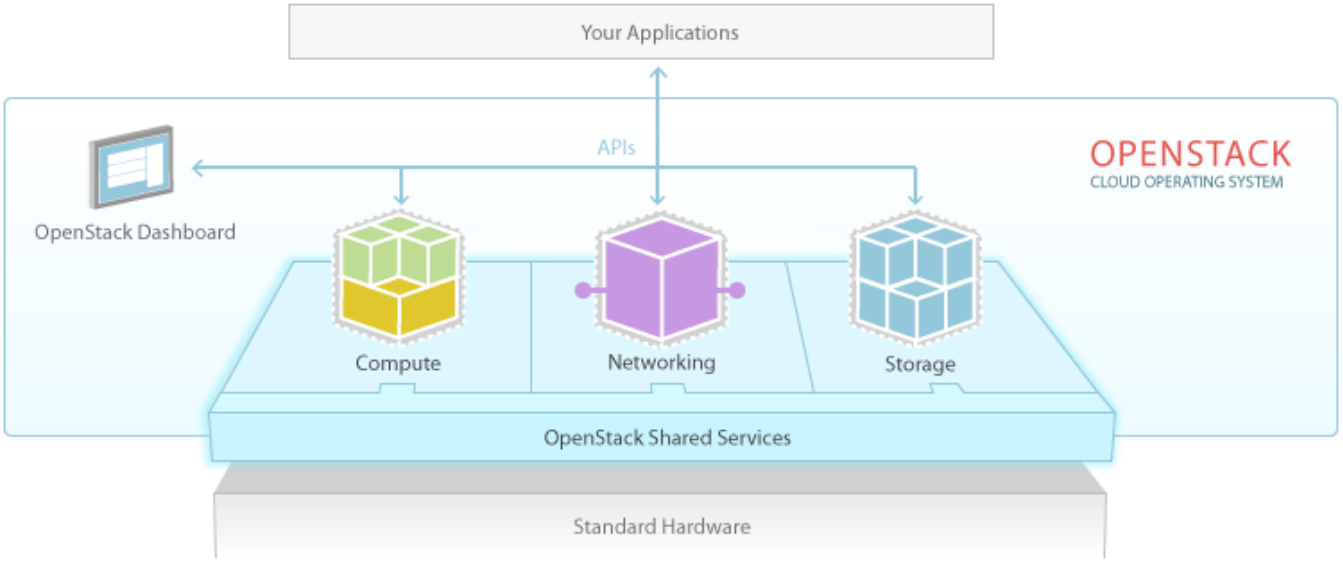
- Simple multi-tiers environment:
  - web application (mediawiki), web servers (apache), Database (MariaDB). And including Memcached.





Case study

- Cloud platform: Openstack



Future work

Future work

- Build a new add-on for LTTng to trace incoming message and outgoing message at the kernel level.
- Track an incoming message within one component.
- Applying proposed approach to generate the execution path of request in a simple web application (web server farm included).
- Build a new plug-in for eclipse to see the request path.
- Implement the approach for the Openstack platform.
- Applying mathematical models (Kalman filter,…) to build a new adaptive sampling mechanism based on the traffic workloads.

Challenges

Challenges

- Support Synchronous/Asynchronous communication between components and within one component: a request can involve execution across threads and process boundaries within one component.
- Collect the events of Linux kernel network stack on one component.
- Simplify the event-schema as compared to Magpie.
- Improve and optimize the solutions for existing problems.

References

- **[1]** Benjamin H. Sigelman, Luiz Andre Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, Chandan Shanbhag, “Dapper, a Large-Scale Distributed Systems Tracing Infrastructure”, Google Technical Report dapper-2010-1, April 2010
- **[2]** Twitter Zipkin. A Distributed Tracing System, https://github.com/twitter/zipkin, 2012.
- **[3]** P. Barham, R. Isaacs, R. Mortier, and D. Narayanan. Magpie: on-line modelling and performance-aware systems. In 9th Workshop on Hot Topics in Operating Systems (HotOS IX), pages 85–90, May 2003.
- **[4]** B. C. Tak, C. Tang, C. Zhang, S. Govindan, B. Urgaonkar, and R. N. Chang, “vPath: precise discovery of request processing paths from blackbox observations of thread and network activities,” in Proceedings of the 2009 onference on USENIX Annual technical conference, 2009, p. 19–19.
- **[5]** E. Koskinen and J. Jannotti, “Borderpatrol: isolating events for black-box tracing,” in ACM SIGOPS Operating Systems Review, 2008, vol. 42, pp. 191–203.
- **[6]** B. Sang, J. Zhan, G. Lu, H. Wang, D. Xu, L. Wang, and Z. Zhang, “Precise, scalable, and online request tracing for multi-tier services of black boxes,” IEEE Transactions on Parallel and Distributed Systems, no. 99, p. 1–1, 2010.

Thank you!

Questions?