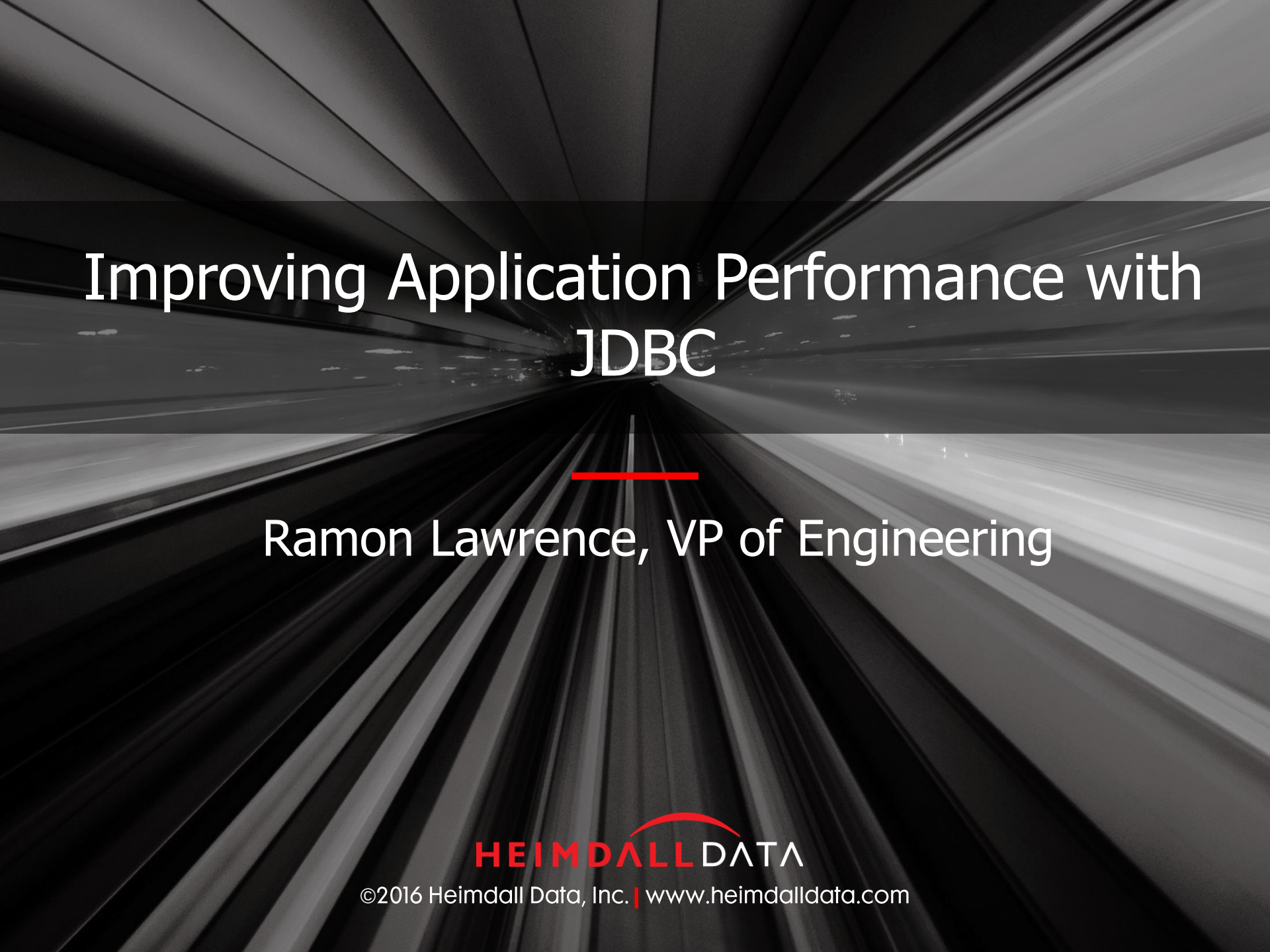


Improving Application Performance with JDBC

Ramon Lawrence, VP of Engineering

HEIMDALL DATA

©2016 Heimdall Data, Inc. | www.heimdalldata.com

Agenda

- Executive Summary
- JDBC Overview
- Comparing existing technologies
- Heimdall Data solution
- Q&A

Copyright © 2016 Heimdall Data, Inc.

Executive Summary

- With IT organizations moving to the cloud, application owners and designers require flexibility to make good decisions.
  - Performance and availability is directly tied to the data layer
  - There is often little visibility into the app-to-database interaction
- Heimdall Data provides application resiliency with:
  - Application performance & scale
  - Application-aware database failover
  - Database security

Copyright © 2016 Heimdall Data, Inc.

Ramon Lawrence

- VP of Engineering, Heimdall Data
- Professor of Computer Science, University of British Columbia
- Founded Unity Data, a data virtualization company for SQL/NoSQL database access
- Built several JDBC drivers for data integration and access to non-relational sources (MongoDB, ServiceNow, Splunk).
- Research on database integration and tiny, embedded databases (LittleD, IonDB) (run in 4K of RAM).

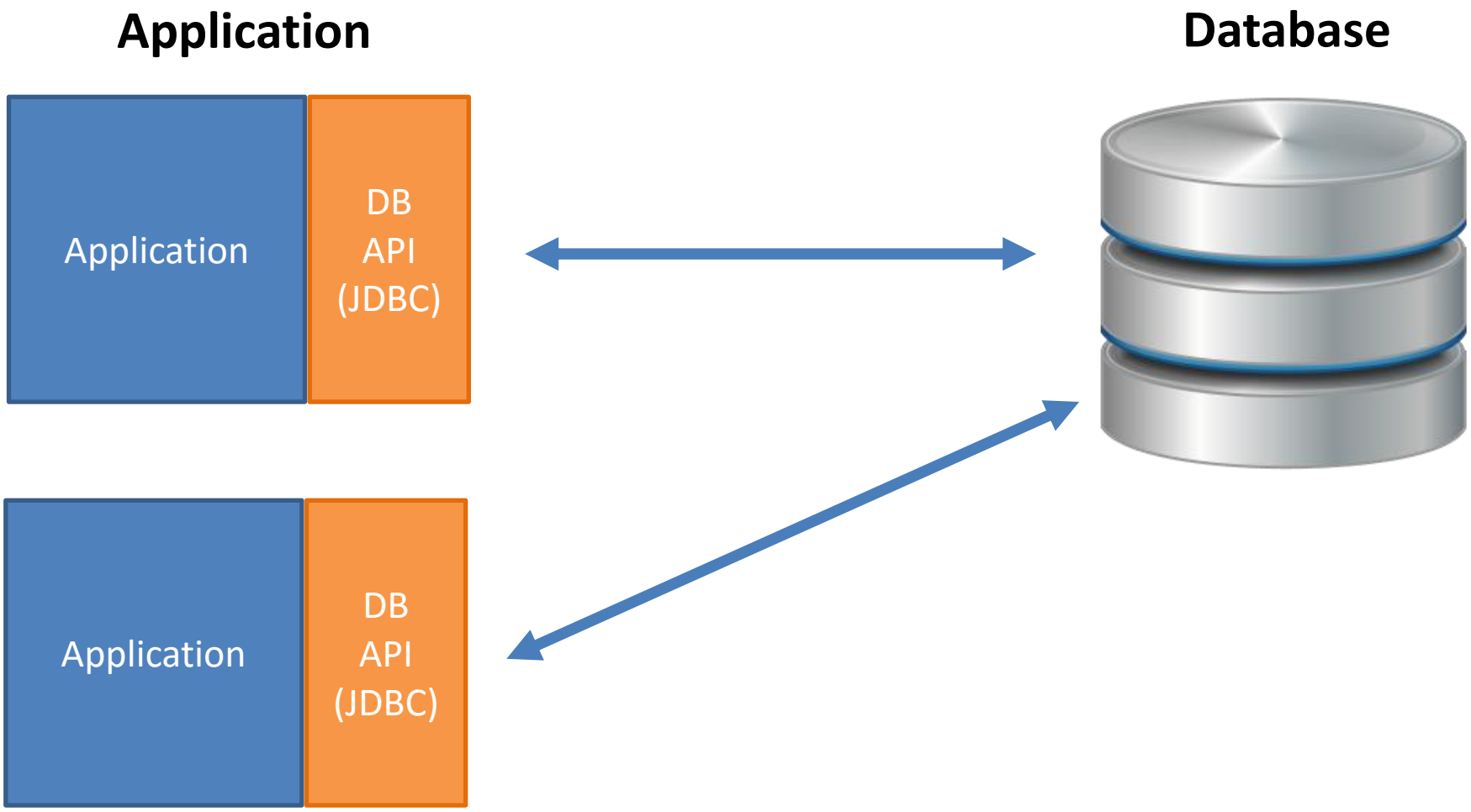
Copyright © 2016 Heimdall Data, Inc.

Who is this for?

- Application developer
  - Reduce development time, especially related to caching and high-availability
- Application owner/architect
  - Have more robust apps that scale and handle failure without architecture changes
- Database administrator
  - Reduce DB load and increase performance
  - Application-level analytics on DB usage

Copyright © 2016 Heimdall Data, Inc. | www.heimdalldata.com

Basic Database and Application Scenario



Copyright © 2016 Heimdall Data, Inc.

©2016 Heimdall Data, Inc. | www.heimdalldata.com

JDBC Overview

- JDBC is an API that contains methods to connect to a database, execute queries, and retrieve results.
- For each database, the vendor writes a JDBC driver that implements the API.
- JVM-based languages (Java, Clojure, Groovy, Scala, JRuby, Jython, ColdFusion) use JDBC for data access.

Copyright © 2016 Heimdall Data, Inc. | www.heimdalldata.com

JDBC Interfaces

- The main interfaces in the JDBC API are:
  - Driver - The main class of the entire driver.
  - Connection - For connecting to the DB.
  - Statement - For executing a query.
  - ResultSet - For storing and manipulating results returned by a Statement.
  - DatabaseMetaData - For retrieving metadata (schema) information from a database.

Copyright © 2016 Heimdall Data, Inc. | www.heimdalldata.com

Key Features of JDBC Drivers

- Key features of a JDBC driver:
  - Create a connection. Close a connection.
  - Execute statements on a connection.
  - Translate Java API calls to network protocol of the database.
  - Transform query results into Java objects.
  - Transactions and PreparedStatements.
- Vendors may not implement all methods in API and may have own custom methods.

Copyright © 2016 Heimdall Data, Inc. | www.heimdalldata.com

JDBC Example


```
try (Connection con = DriverManager.getConnection(url, uid, pw);
     Statement stmt = con.createStatement(); )
{
    ResultSet rst = stmt.executeQuery("SELECT ename,salary
                                       FROM Emp");
    System.out.println("Employee Name,Salary");
    while (rst.next())
        System.out.println(rst.getString("ename")
                           +","+rst.getDouble("salary"));
}
catch (SQLException ex)
{
    System.err.println(ex);
}
```

Declare resources

Statement and Connection objects closed by end of try

Copyright © 2016 Heimdall Data, Inc. | www.heimdalldata.com

JDBC Prepared Statements Example


```
try (Connection con = DriverManager.getConnection(url, uid, pw);
     PreparedStatement pstmt =
          con.prepareStatement("SELECT ename,salary FROM Emp"
               " WHERE salary > ? AND ename LIKE ?");
     )
{
   pstmt.setDouble(1, 30000);
   pstmt.setString(2, "%e%");
   ResultSet rst = pstmt.executeQuery();
   System.out.println("Employee Name,Salary");
   while (rst.next())
   {   System.out.println(rst.getString("ename")
                                   +","+rst.getDouble("salary"));
   }
   rst.close();
}
catch (SQLException ex) { … }
```

Copyright © 2016 Heimdall Data, Inc.

Question: How many of the following are JDBC objects?

- Driver
- Connection
- HashMap
- ResultSet
- Statement

Copyright © 2016 Heimdall Data, Inc.

©2016 Heimdall Data, Inc. | www.heimdalldata.com

JDBC is "Low-Level"

- JDBC is considered a low-level API for database access.
  - Often repetitive code
  - Need to carefully handle exceptions and closing of resources
  - Developers must understand SQL and relational querying

- **Fundamental challenge:** Relational databases represent data as tables with rows while Java programmers wants to interact with classes and object instances.
  - Translation back and forth between tables and objects is tedious, error-prone, and time-consuming.

Copyright © 2016 Heimdall Data, Inc. | www.heimdalldata.com

Object-Relational Mapping

- **Object-relational mapping** (**ORM**) libraries simplify this mapping which allows application developer more time to focus on fundamental program logic/features.
  - Java Persistence Architecture (JPA) has been developed as a standard interface.
  - Various systems (e.g. Hibernate) perform ORM and follow JPA.

Copyright © 2016 Heimdall Data, Inc. | www.heimdalldata.com

ORM Example


```
EntityManagerFactory factory =
            Persistence.createEntityManagerFactory("workson");
EntityManager em = factory.createEntityManager();

Query q = em.createQuery("SELECT e FROM Employee e");

for (Object obj : q.getResultList())
{   Employee e = (Employee) obj;
    System.out.println(e.getName() + "," + e.getSalary());
}

em.close();
factory.close();
```

Copyright © 2016 Heimdall Data, Inc.

ORM Example (Mapping)


```
@Entity
@Table(name="Emp")
public class Employee {
    @Id
    @Column(name="eno")
    private String number;

    @Column(name="ename")
    private String name;

    @Column(name="title")
    private String title;

    @Column(name="salary")
    private double salary;

    @ManyToOne
    @JoinColumn(name="dno")
     private Department department;
```

Copyright © 2016 Heimdall Data, Inc. | www.heimdalldata.com

ORM Advantages/Disadvantages

Advantages:

- Hides database organization
- Consistent Java interface
- Often allows auto-generation of SQL
- Object-level caching
- Higher-level API

Disadvantages:

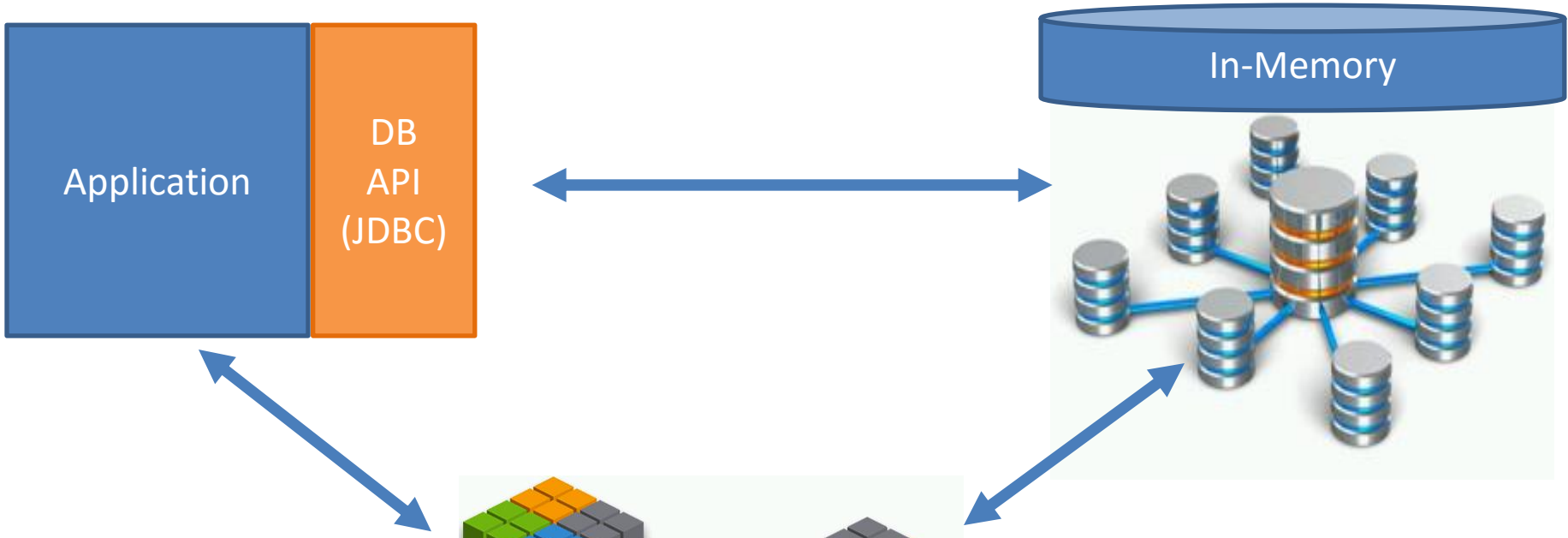
- Another layer on top of JDBC to manage
- Auto-generation of SQL may not be ideal or high-performing even with caching
- Cannot be easily added to an existing application

Copyright © 2016 Heimdall Data, Inc.

Challenge: In-Memory Performance/Scale-out

Application

Database



Data Grid Cache

Challenges:

- More systems/hardware
- Keeping cache in-sync with DB

Copyright © 2016 Heimdall Data, Inc.

Data layer scale/performance application issues

- Stateless application servers are easier to scale than consistent databases.

- Scaling the data layer is often:
  - Costly especially for commercial licensed DBs
  - Challenging to implement/deploy/manage
  - Results in compromises/trade-offs of consistency, performance, and availability

Copyright © 2016 Heimdall Data, Inc.

Application Performance Improvements

- Applications can improve performance when interacting with the data layer by:
  - Making more precise/efficient requests (SQL)
  - Minimizing the amount of data requested (caching, DB offloading)
  - Batching requests to avoid network latency

Copyright © 2016 Heimdall Data, Inc.

High Availability and Maximum Up-Time

- Application layer achieves high availability by replication of stateless servers
- Data layer achieves high availability by redundancy of systems and techniques for replication and fail-over
- Applications should be hidden from data layer failures and transparently adapt to changes

Copyright © 2016 Heimdall Data, Inc.

Intelligent JDBC Drivers

- **Idea:** Add more capabilities at the JDBC driver level (inside the application) for simplicity and performance.

Copyright © 2016 Heimdall Data, Inc.

Example #1: JDBC Drivers for NoSQL Sources

- Drivers for MongoDB, Splunk, ServiceNow
  - Translate SQL to native dialect
  - Handle features not supported by data layer
- Example:
  - JDBC driver for Couchbase built with Simba Technologies
  - JDBC API is flexible to handle non-relational sources while maintaining standards compliance with other software

Copyright © 2016 Heimdall Data, Inc.

Example #2: Heimdall Data

Customer Challenges

- **Application owners have little visibility at the data access layer**
- DBA's need better application performance visibility
  - Inefficient SQL queries
  - Network latency

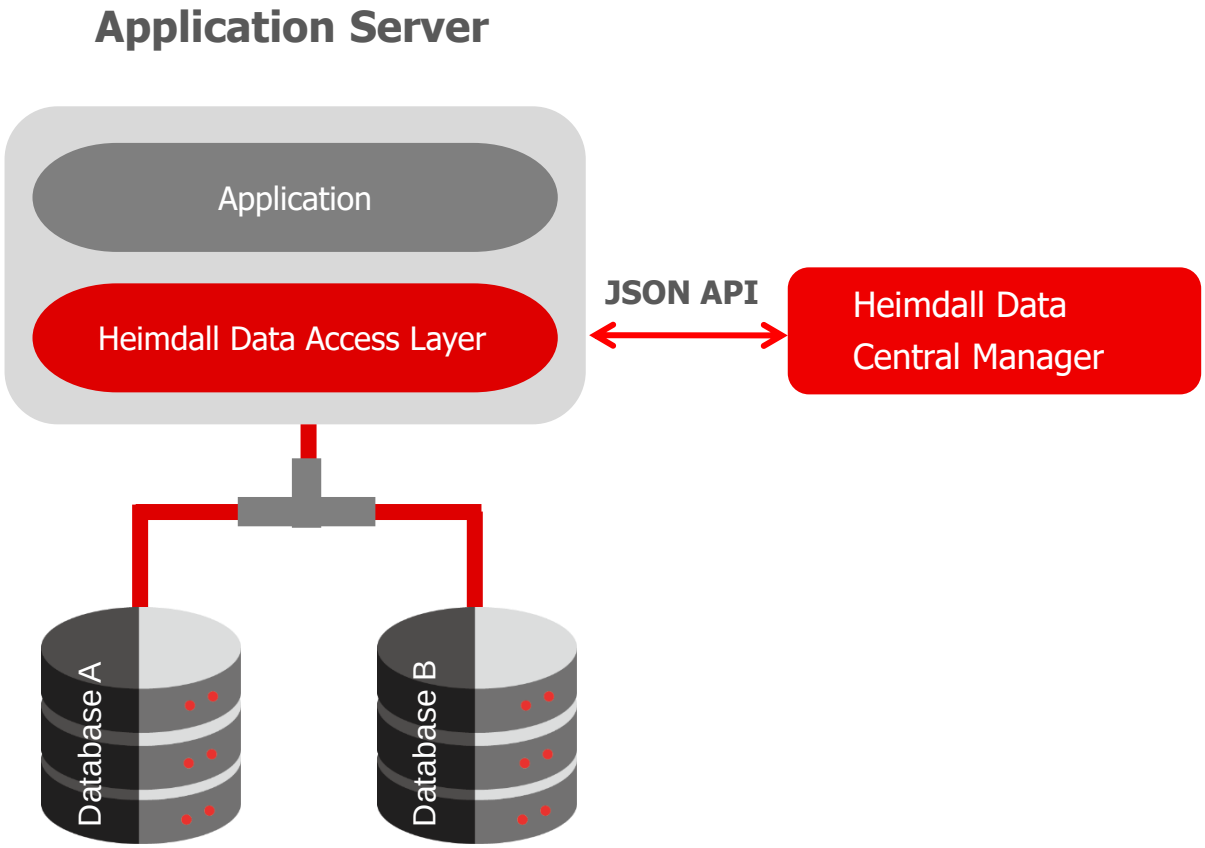
Heimdall Data **transparently** improve application performance through data access optimization.

- Visibility: SQL performance from an *application* perspective.
- Reliability: Application aware HA failover
- Performance: Application optimization with transparent caching
- Transparency: Requires **ZERO** application level code changes

Copyright © 2016 Heimdall Data, Inc.

Heimdall Data Resiliency Platform

- Analytics & Monitoring
- SQL Caching & Optimizations
- Automated HA DB Failover
- Query Transformation
- Query Routing
- Data Masking & Security



Copyright © 2016 Heimdall Data, Inc.

Heimdall Resiliency Use Cases

- High Performance
  - Intelligent distributed SQL caching
- High Availability
  - Application isolated from data layer failures at the JDBC connection level
- Security
  - Configurable query filtering and pattern matching to protect against common data attacks (SQL injection)
  - Learning mode with recommendations

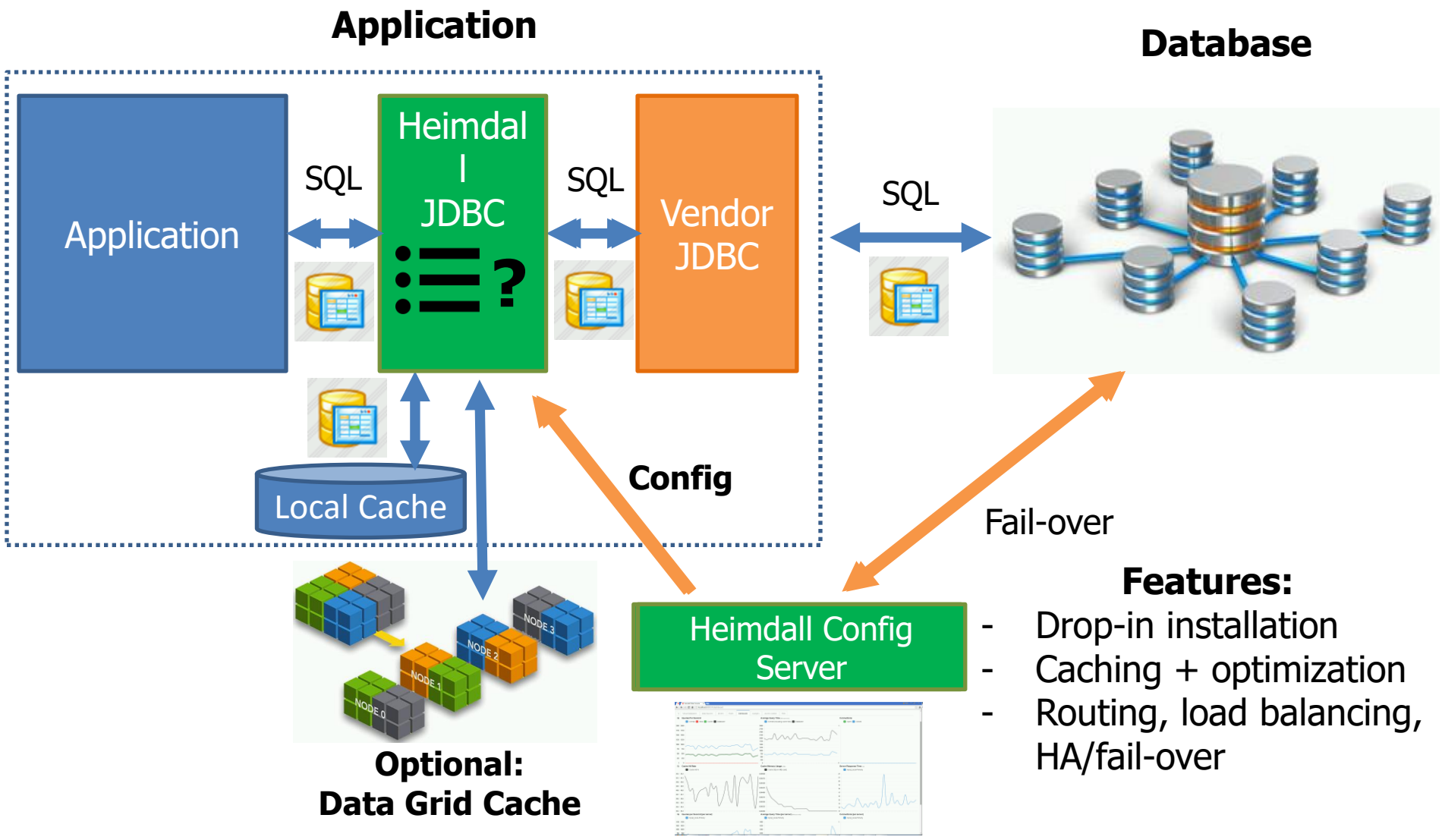
Copyright © 2016 Heimdall Data, Inc.

Transparent Caching

- Adding caching layer often requires code changes (complex, costly).
- Heimdall caches at JDBC query/ResultSet level.
- Heimdall provides recommendations on what to cache (the hardest design/analysis problem).

Copyright © 2016 Heimdall Data, Inc.

Heimdall Caching Example



Features:

- Drop-in installation
- Caching + optimization
- Routing, load balancing, HA/fail-over

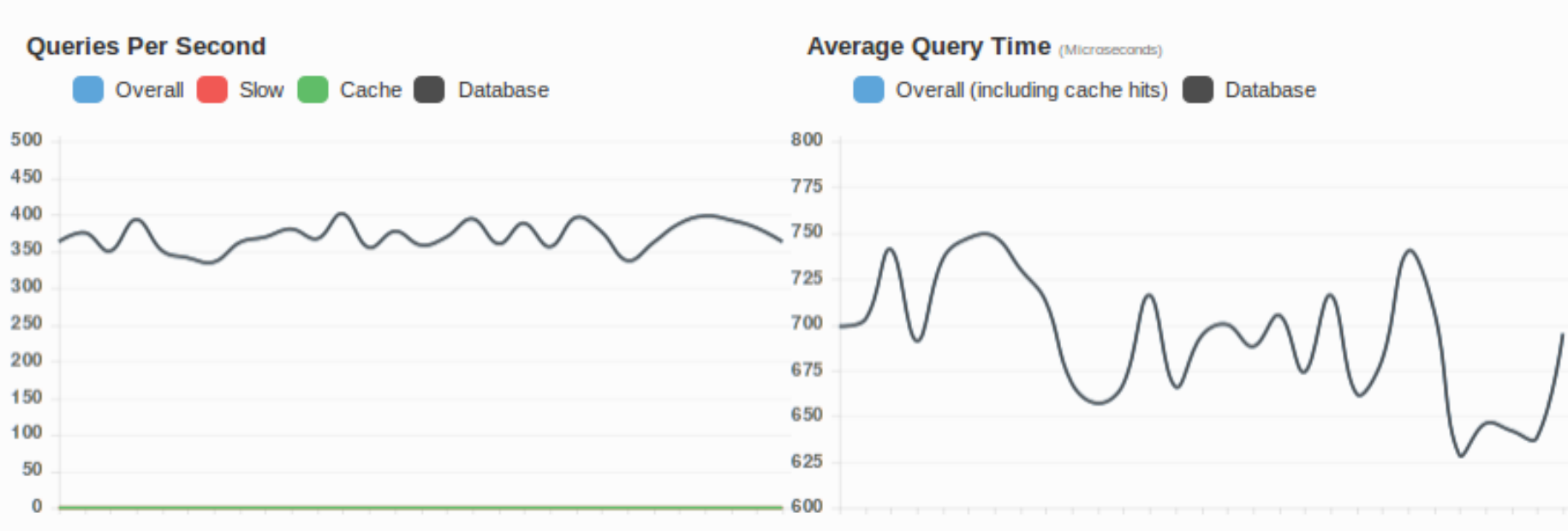
Copyright © 2016 Heimdall Data, Inc.

Example Setup

- JIRA with JIRA benchmark traffic
- Two PostgreSQL databases setup in primary-secondary replication

Copyright © 2016 Heimdall Data, Inc.

Query Analytics - Dashboard



Copyright © 2016 Heimdall Data, Inc.

Cache Recommendations

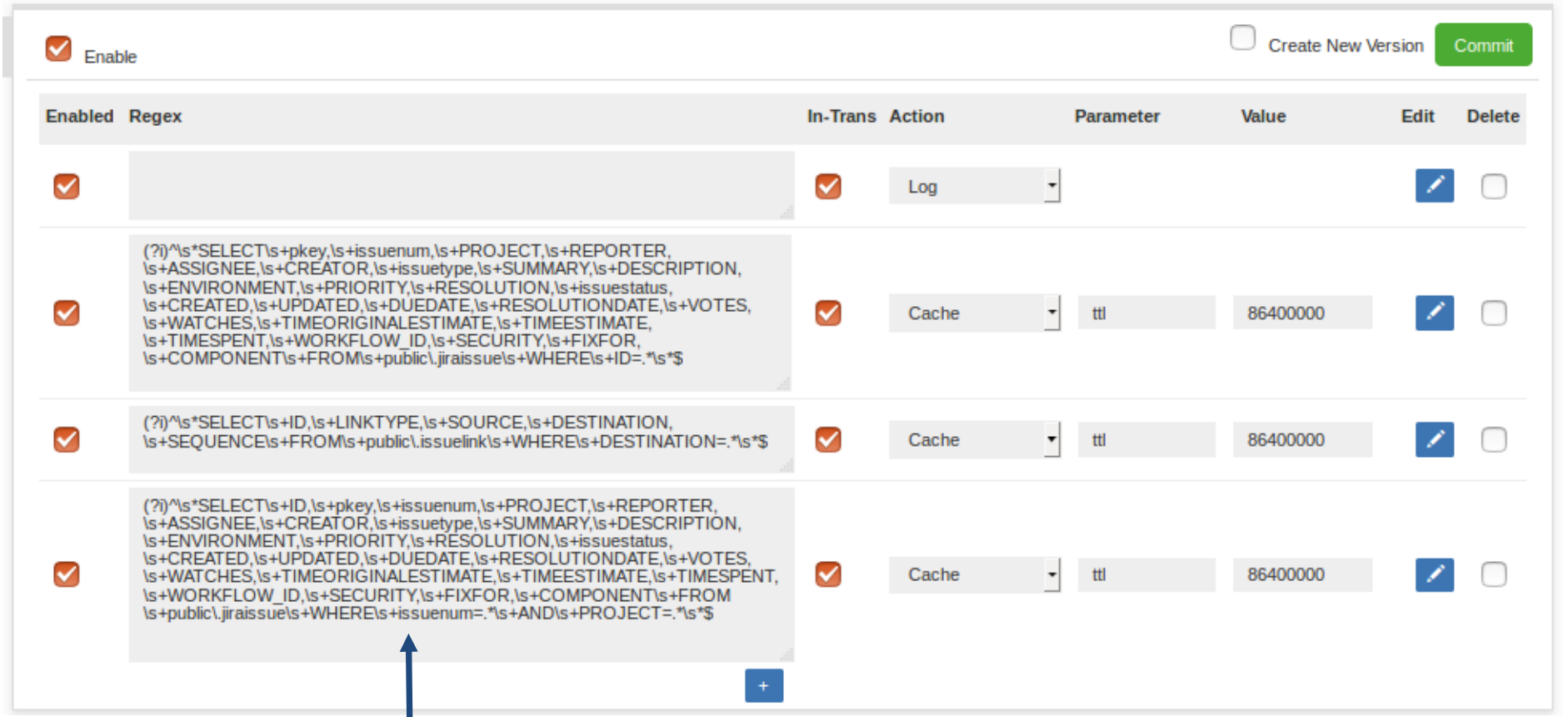
22% Server Time
80 Cache Safety

2000+ executions
0.75 ms/query

Rank		Query	Server Time (%)	Duplicate Query & Response (%)	Cache Safety Rating	Cache Time (s)	Cache Hit (%)	Count	Query Response Time (µs)	Result Retrieval Time (µs)	Total KBs
1	Cache	SELECT pkey, issuenum, PROJECT, REPORTER, ASSIGNEE, CREATOR, issuetype, SUMMARY, DESCRIPTION, ENVIRO NMENT, PRIORITY, RESOLUTION, issuestatus, CREATED, UPDATED, DUEDATE, RESOLUTIONDATE, VOTES, WATCHES, TIMEORIGINALESTIMATE, TIMEESTIMATE, TIMESPENT, WORKFLOW_ID, SECURITY, FIXFOR, COMPONENT FROM public .jiraissue WHERE ID=?	22	100	80	86400	0.0	2175	732.5	761.9	2336
2	Cache	SELECT ID, LINKTYPE, SOURCE, DESTINATION, SEQUENCE FROM public.issuelink WHERE DESTINATION=?	8	60	0	86400	0.0	919	638.2	649.6	919
3	Cache	SELECT ID, pkey, issuenum, PROJECT, REPORTER, ASSIGNEE, CREATOR, issuetype, SUMMARY, DESCRIPTION, EN VIRONMENT, PRIORITY, RESOLUTION, issuestatus, CREATED, UPDATED, DUEDATE, RESOLUTIONDATE, VOTES, WATC HES, TIMEORIGINALESTIMATE, TIMEESTIMATE, TIMESPENT, WORKFLOW_ID, SECURITY, FIXFOR, COMPONENT FROM pu blic.jiraissue WHERE issuenum=? AND PROJECT=?	7.5	0	0	86400	0.0	612	904.0	952.1	658
4	Cache	SELECT ID, LINKTYPE, SOURCE, DESTINATION, SEQUENCE FROM public.issuelink WHERE SOURCE=?	6.4	91	0	86400	0.0	671	697.7	710.3	674
5	Cache	SELECT ID, filename, contenttype, avatartype, owner, systemavatar FROM public.avatar WHERE filename= ? AND avatartype=? AND systemavatar=?	5.9	50	0	86400	0.0	613	708.2	726.4	613
6		INSERT INTO public.userhistoryitem (ID, entitytype, entityid, USERNAME, lastviewed, data) VALUES (?, ?, ?, ?, ?, ?)	5.8	100	0	86400	0.0	599	710.2	710.2	599
7	Cache	SELECT SOURCE_NODE_ID, SOURCE_NODE_ENTITY, SINK_NODE_ID, SINK_NODE_ENTITY, ASSOCIATION_TYPE, SEQUENC E FROM public.nodeassociation WHERE SOURCE_NODE_ID=? AND SOURCE_NODE_ENTITY=? AND SINK_NODE_ENTITY=? AND ASSOCIATION_TYPE=?	5.6	100	60	86400	0.0	612	677.3	685.4	612
8	Cache	SELECT ID, ISSUE, CUSTOMFIELD, PARENTKEY, STRINGVALUE, NUMBERVALUE, TEXTVALUE, DATEVALUE, VALUETYPE FROM public.customfieldvalue WHERE ISSUE=?	4	0	0	86400	0.0	306	965.9	1028.3	490
9	Cache	SELECT ID, issueid, MIMETYPE, FILENAME, CREATED, FILESIZE, AUTHOR, zip, thumbnailable FROM public.fi leattachment WHERE issueid=? ORDER BY FILENAME ASC, CREATED DESC	3.9	100	50	86400	0.0	307	924.0	939.5	307
10	Cache	select c.id, c.cname, c.description, c.lead, c.assigneetype, c.project from public.nodeassociation n a join public.component c on c.id = na.sink_node_id where na.source_node_entity = ? and na.source_no de_id = ? and na.sink_node_entity = ? and na.association_type = ?	3.5	100	50	86400	0.0	306	847.7	851.9	306
11	Cache	SELECT ID, actionlevel, rolelevel FROM public.jiraaction WHERE issueid=? AND actiontype=? ORDER BY C REATED DESC	3.2	0	0	86400	0.0	307	775.0	801.8	307
12	Cache	SELECT SOURCE_NAME, SINK_NODE_ID, SINK_NODE_ENTITY, ASSOCIATION_TYPE, SEQUENCE, CREATED FROM public. userassociation WHERE SINK_NODE_ID=? AND SINK_NODE_ENTITY=? AND ASSOCIATION_TYPE=?	3.1	100	50	86400	0.0	307	748.3	758.7	307
13	Cache	SELECT ID, ISSUEID, GLOBALID, TITLE, SUMMARY, URL, ICONURL, ICONTITLE, RELATIONSHIP, RESOLVED, STATU SNAME, STATUSDESCRIPTION, STATUSICONURL, STATUSICONTITLE, STATUSICONLINK, STATUSCATEGORYKEY, STATUSC ATEGORYCOLORNAME, APPLICATIONTYPE, APPLICATIONNAME FROM	2.9	100	50	86400	0.0	307	704.2	712.3	307

Copyright © 2016 Heimdall Data, Inc.

Caching Rules



Rules are regular expressions

Copyright © 2016 Heimdall Data, Inc.

Query Analytics - Cache Benefits

11% Server Time
50% reduction

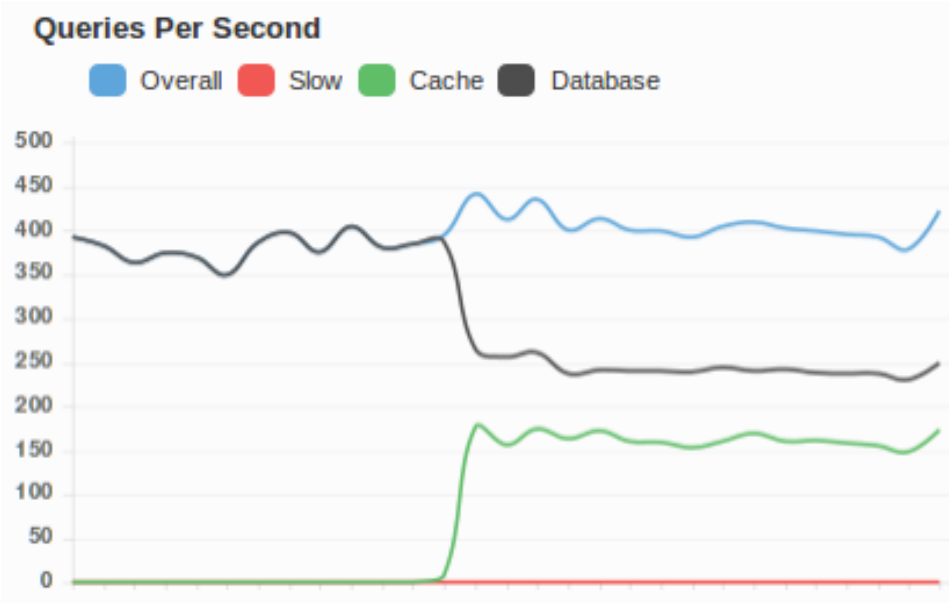
0.1 ms/query
8x faster

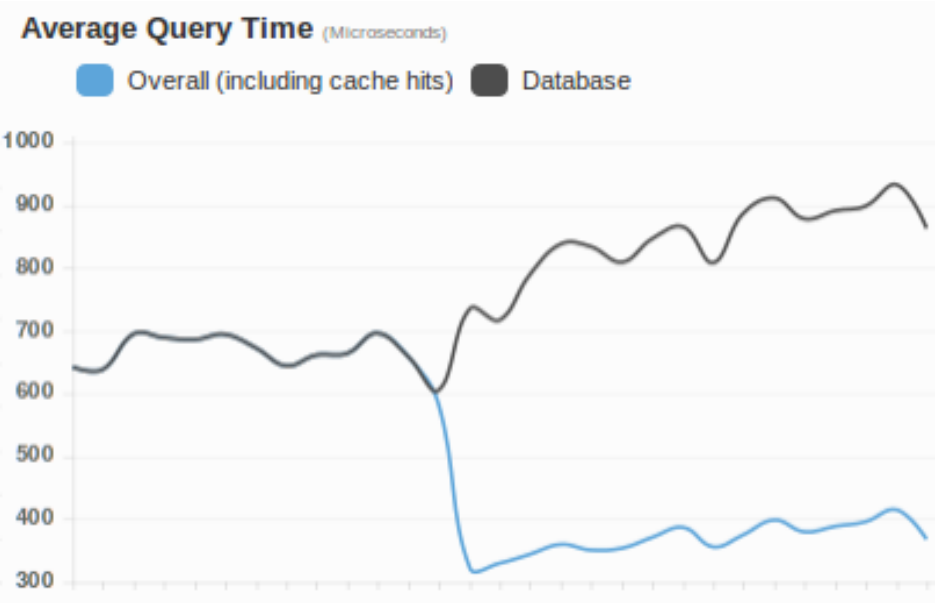
Rank		Query	Server Time (%)	Duplicate Query & Response (%)	Cache Safety Rating	Cache Time (s)	Cache Hit (%)	Count	Query Response Time (µs)	Result Retrieval Time (µs)	Total KBs
1	Cache	SELECT pkey, issuenum, PROJECT, REPORTER, ASSIGNEE, CREATOR, issuetype, SUMMARY, DESCRIPTION, ENVIRO NMENT, PRIORITY, RESOLUTION, issuestatus, CREATED, UPDATED, DUEDATE, RESOLUTIONDATE, VOTES, WATCHES, TIMEORIGINALESTIMATE, TIMEEST MATE, TIMESPENT, WORKFLOW_ID, SECURITY, FIXFOR, COMPONENT FROM public .jiraissue WHERE ID=?	11	100	90	86400	80.6	4724	127.6	142.0	5193
2	Cache	SELECT ID, pkey, issuenum, PROJECT, REPORTER, ASSIGNEE, CREATOR, issuetype, SUMMARY, DESCRIPTION, EN VIRONMENT, PRIORITY, RESOLUTION, issuestatus, CREATED, UPDATED, DUEDATE, RESOLUTIONDATE, VOTES, WATC HES, TIMEORIGINALESTIMATE, TIMEESTIMATE, TIMESPENT, WORKFLOW_ID, SECURITY, FIXFOR, COMPONENT FROM pu blic.jiraissue WHERE issuenum=? AND PROJECT=?	9.5	0	0	86400	50.0	1317	379.3	406.5	1457
3	Cache	SELECT ID, ISSUE, CUSTOMFIELD, PARENTKEY, STRINGVALUE, NUMBERVALUE, TEXTVALUE, DATEVALUE, VALUETYPE FROM public.customfieldvalue WHERE ISSUE=?	8.8	0	0	86400	0.2	659	707.9	747.7	1038
4	Cache	SELECT SOURCE_NODE_ID, SOURCE_NODE_ENTITY, SINK_NODE_ID, SINK_NODE_ENTITY, ASSOCIATION_TYPE, SEQUENC E FROM public.nodeassociation WHERE SOURCE_NODE_ID=? AND SOURCE_NODE_ENTITY=? AND SINK_NODE_ENTITY=? AND ASSOCIATION_TYPE=?	7.6	100	70	86400	0.2	1318	302.9	308.2	1318
5	Cache	select c.id, c.cname, c.description, c.lead, c.assigneetype, c.project from public.nodeassociation n a join public.component c on c.id = na.sink_node_id where na.source_node_entity = ? and na.source_no de_id = ? and na.sink_node_entity = ? and na.association_type = ?	7	100	60	86400	0.2	659	563.8	567.0	659
6	Cache	SELECT ID, issueid, MIMETYPE, FILENAME, CREATED, FILESIZE, AUTHOR, zip, thumbnailable FROM public.fi leattachment WHERE issueid=? ORDER BY FILENAME ASC, CREATED DESC	7	100	60	86400	0.2	659	556.7	567.5	659
7	Cache	SELECT SOURCE_NAME, SINK_NODE_ID, SINK_NODE_ENTITY, ASSOCIATION_TYPE, SEQUENCE, CREATED FROM public. userassociation WHERE SINK_NODE_ID=? AND SINK_NODE_ENTITY=? AND ASSOCIATION_TYPE=?	6.4	100	60	86400	0.2	660	507.6	516.5	660
8	Cache	SELECT ID, LINKTYPE, SOURCE, DESTINATION, SEQUENCE FROM public.issuelink WHERE SOURCE=?	5.8	92	0	86400	54.8	1457	210.6	219.0	1460
9	Cache	SELECT ID, actionlevel, rolelevel FROM public.jiraaction WHERE issueid=? AND actiontype=? ORDER BY C REATED DESC	5.3	0	0	86400	0.2	659	424.8	444.5	659

Server Time (%)	Duplicate Query & Response (%)	Cache Safety Rating	Cache Time (s)	Cache Hit (%)	Count	Query Response Time (µs)	Result Retrieval Time (µs)	Total KBs
11	100	90	86400	80.6	4724	127.6	142.0	5193

Copyright © 2016 Heimdall Data, Inc.

Query Analytics - Benefit of Caching





Cached queries reduce DB load by about 50%

Average query time reduced by 50% (2 times overall).

Copyright © 2016 Heimdall Data, Inc.

Caching - How it Works

- Cache implementation is flexible (local cache, Hazelcast, redis, memcached).
  - Operates as a look-aside cache
- Cache recommendations are built based on analyzing query responses and detecting duplication.
  - Uses history of query answers to determine frequency that data is changing
  - Highly transactional (update) tables are detected as not cacheable.

Copyright © 2016 Heimdall Data, Inc. | www.heimdalldata.com

Caching - The Hard Questions

- How does cache stay in-sync with database?
  - Heimdall invalidates cache entries when detects INSERT/UPDATE/DELETE that may change cache entry (done at table level).
  - Changes that do not flow through Heimdall:
    - May call API to trigger invalidation
    - May use trigger-based invalidation at the database

Copyright © 2016 Heimdall Data, Inc.

Caching - The Hard Questions (cont.)

- How does cache at each application server stay in sync?
  - Depends on cache implementation. Heimdall supports local cache (no sync) or grid cache like Hazelcast which handles synchronization.

Copyright © 2016 Heimdall Data, Inc.

SQL Caching versus Object Caching

- Object caching at the app level:
  - Allows developer full control
  - Caching at higher level after objects are built
  - May be done in ORM framework
- SQL result caching at the driver level:
  - Can be done with no code changes
  - More broad range of cache possibilities as sees all traffic
  - Benefit even if implement object caching

Copyright © 2016 Heimdall Data, Inc.

Asynchronous Updates

- Async updates useful for logging to not slow down page generation for stats tracking.
- Logging takes time:

1 ms/insert

Rank		Query	Server Time (%)	Duplicate Query & Response (%)	Cache Safety Rating	Cache Time (s)	Cache Hit (%)	Count	Query Response Time (µs)	Result Retrieval Time (µs)	Total KBs
1		INSERT INTO public.userhistoryitem (ID, entitytype, entityid, USERNAME, lastviewed, data) VALUES (?, ?, ?, ?, ?, ?)	90	100	0	86400	0.0	567	1008.0	1008.0	567
2		DELETE FROM public.userhistoryitem WHERE (USERNAME = ?) AND (entitytype = ?) AND (entityid IN (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?))	8.3	100	0	86400	0.0	52	1020.6	1020.6	52
3	Cache	SELECT ID, entitytype, entityid, USERNAME, lastviewed, data FROM public.userhistoryitem WHERE entity type=? AND entityid=? AND USERNAME=?	1.1	14	0	86400	0.0	7	992.6	1016.1	7
4		UPDATE public.userhistoryitem SET entitytype=?, entityid=?, USERNAME=?, lastviewed=?, data=? WHERE I D=?	1.1	100	0	86400	0.0	7	986.4	986.4	7

Copyright © 2016 Heimdall Data, Inc.

Asynchronous Updates - Benefits

0.05 ms/insert

Rank		Query	Server Time (%)	Duplicate Query & Response (%)	Cache Safety Rating	Cache Time (s)	Cache Hit (%)	Count	Query Response Time (µs)	Result Retrieval Time (µs)	Total KBs
1		INSERT INTO public.userhistoryitem (ID, entitytype, entityid, USERNAME, lastviewed, data) VALUES (?, ?, ?, ?, ?, ?)	54	100	0	86400	100.0	563	47.6	47.6	563
2	Cache	SELECT ID, entitytype, entityid, USERNAME, lastviewed, data FROM public.userhistoryitem WHERE entity type=? AND entityid=? AND USERNAME=?	43	9.1	0	86400	0.0	11	1965.1	1987.3	11
3		DELETE FROM public.userhistoryitem WHERE (USERNAME = ?) AND (entitytype = ?) AND (entityid IN (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?))	2.6	100	0	86400	100.0	51	25.2	25.2	51
4		UPDATE public.userhistoryitem SET entitytype=?, entityid=?, USERNAME=?, lastviewed=?, data=? WHERE I D=?	0	100	0	86400	100.0	11	34.9	34.9	11

Copyright © 2016 Heimdall Data, Inc.

High Availability and Fail-over

- Application-aware fail-over (especially useful for relational open source MySQL/PostgreSQL)
- **Detect, trigger, adapt**
- Query holding
- JDBC connections often not affected by database failures from app perspective

Copyright © 2016 Heimdall Data, Inc.

Routing, Load Balancing, Transformation

- Based on query patterns can:
  - Route SQL to different servers (including different vendors) – migrations, development
  - Dynamic load balance across machines (read/write, read-only).
  - Transform SQL into different SQL (migrations, quick SQL fixes in code)

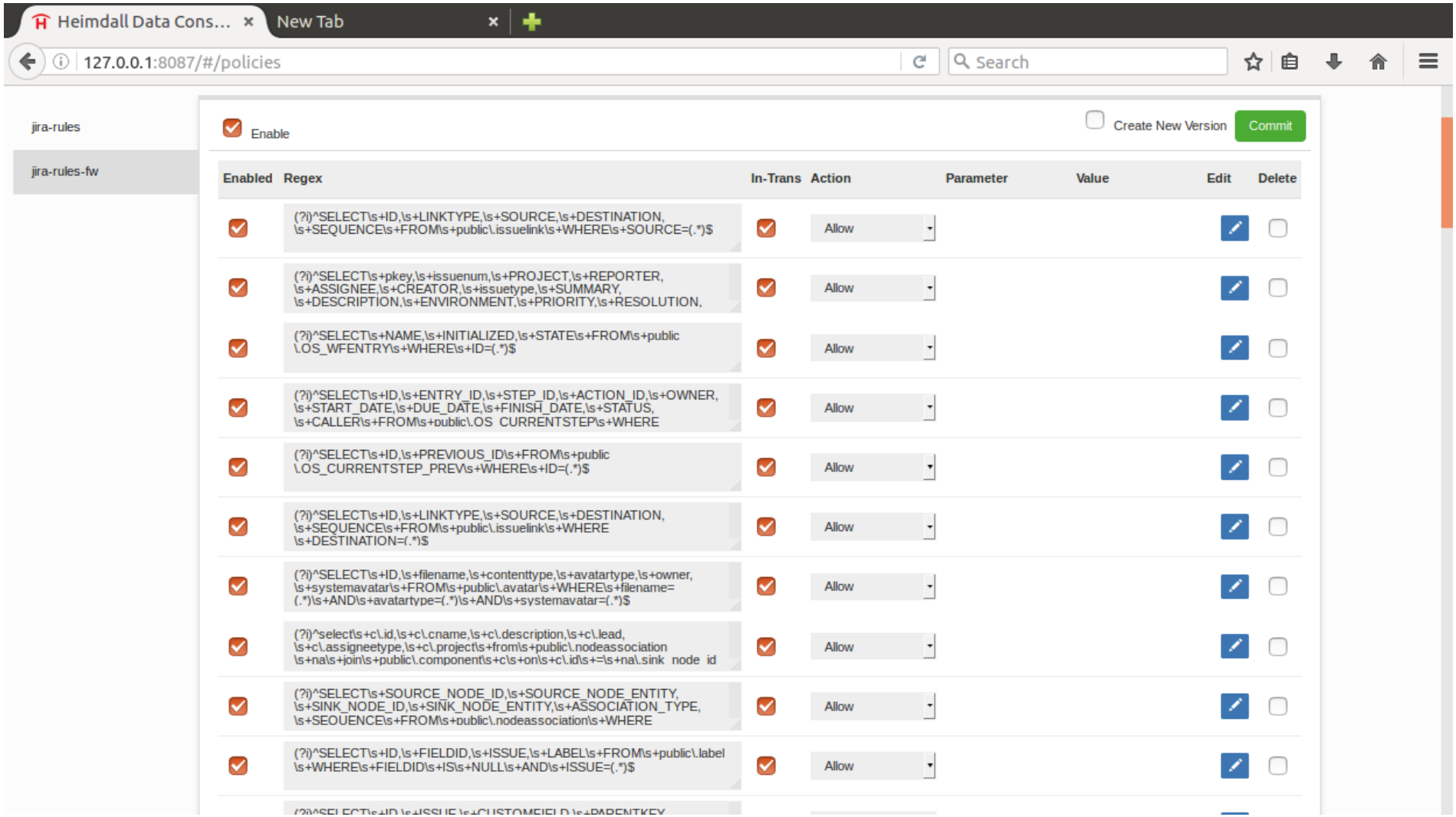
Copyright © 2016 Heimdall Data, Inc.

SQL Security

- Heimdall's rule engine allows for enforcement of security constraints at the SQL query level
  - Query pattern filtering (deny, log)
  - Identifies known query patterns (whitelist) automatically
  - Provides ***audit trail*** on all SQL requests that is database independent

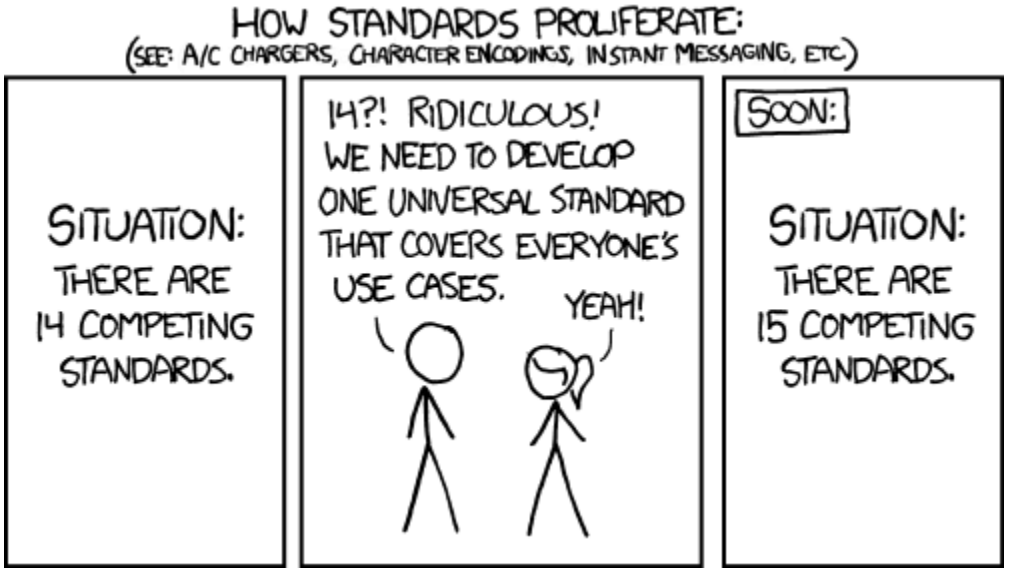
Copyright © 2016 Heimdall Data, Inc.

SQL Security Learned Rules



Copyright © 2016 Heimdall Data, Inc. | www.heimdalldata.com

Looking to the Future



https://xkcd.com/927/

- Avoid API Proliferation!
- Abstract other interfaces for similar resilience benefits
  - HTTP (URLConnection)
  - LDAP (LDAPConnection)
  - DNS (InetAddress)
  - NoSQL Interfaces (Mongo, Cassandra)

Copyright © 2016 Heimdall Data, Inc.

Conclusions

- Heimdall JDBC driver allows for applications to:
  - Increase performance by adding caching layer with no code changes
  - Be resilient to failures in the data layer
  - Be flexible for data layer changes (load balancing, cloud deployments, routing)
  - Identify and block potential SQL attacks before they are sent to the database

Copyright © 2016 Heimdall Data, Inc. | www.heimdalldata.com

©2016 Heimdall Data, Inc. | www.heimdalldata.com