Healthcare Outbounds: Unbounded Scalability

Robert Stephens, Director of Revenue Cycle Development
Sujoy Acharya, Director & Principal Engineer
TalkConfiguration<Integer, String> agenda

agenda.put(1, “The Problem Statement”);
agenda.put(2, “v1. High level Design”);
agenda.put(3, “SP Results”);
agenda.put(4, “v2. Redesigning the system with Apache Ignite”);
agenda.put(5, “v3. Refactoring”);
agenda.put(6, “Q/A”);
Outbound

The Problem Statement
Performance Bottleneck

System updates need to be echoed out to other systems in real time. As these updates need to be sent in sequence, the time spent transforming the data into output format becomes a bottleneck for clients with large volume.

Each echo contains a large quantity of patient data.

In addition to being able to handle a peak period load, the system must be able to recover and backload due to an outage in a reasonable timeframe.
Background

Healthcare revenue cycle product used in United States and global market

Includes Scheduling, Registration, Patient Accounting

The registration ADT (Admission, Discharge, Transfer) transaction data needs to be sent out to other systems in HL7 format

Java platform with horizontal scaling

Challenged to scale the outbound interface for larger clients
Options

Tune code

Scale vertical (Bigger Servers)

Execute transformation sooner

Process in parallel.
This approach seemed best
High Level Design

V1. Database Driven Design
Initial Design
## Initial Design

<table>
<thead>
<tr>
<th>Seq No</th>
<th>TransactionID</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>t001</td>
<td>bed01</td>
</tr>
<tr>
<td>2</td>
<td>t002</td>
<td>bed999</td>
</tr>
<tr>
<td>3</td>
<td>t003</td>
<td>bed01</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TransactionID</th>
<th>Key1</th>
<th>...</th>
<th>Key12</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>t001</td>
<td>bed01</td>
<td></td>
<td></td>
<td>Processing</td>
</tr>
<tr>
<td>t002</td>
<td>bed999</td>
<td></td>
<td></td>
<td>Processing</td>
</tr>
<tr>
<td>t003</td>
<td>bed01</td>
<td></td>
<td></td>
<td>Processing</td>
</tr>
</tbody>
</table>
Processing Rate

65000
Re-Designing

In-memory Architecture
@DataAccessPoints()

Insert → Check → Remove → Update
@Cacheable?
new SqlQuery(Check, "Dependency")

String queryString = String.format("FROM MESSAGEKEY WHERE KEY in (%s)", commaSeparatedKeys);
SqlQuery<Long, MessageKey> igniteQuery = new SqlQuery<>((MessageKey.class, queryString));
igniteQuery.setLocal(true);

List<Entry<Long, MessageKey>> matchingKeys = messageKeyCache.query(igniteQuery).getAll();
boolean hasDependency = matchingKeys != null && matchingKeys.size() > 0;
new InsertQuery(Dependency, "Status")

```java
List<String> keys = newTransactionMessage.getKeysAsString();
String transactionStatus = hasDependency(keys) ? WAITING : PROCESSING;
logger.debug(String.format("Keys are %s and Status %s", keys, transactionStatus));
newTransactionMessage.setStatus(transactionStatus);
/**
 * Insert in message status and key
 */
messageStsCache.put(newTransactionMessage.getJmsId(), newTransactionMessage);
newTransactionMessage.getKeys().forEach(key -> {
    messageKeyCache.put(key.getId(), key);
});
```
new RemoveQuery(Release, "Dependant")
new Result(Metrics)

```java
long startTime = System.nanoTime();
hasDependencies = messageStatusDao.insertMessage(msDto);
long endTime = System.nanoTime();

long timeSpentInDb = endTime - startTime;

startTime = System.nanoTime();
hasDependencies = messageStatusCacheableDao.insertMessage(msDto);
endTime = System.nanoTime();

long timeSpentInCache = endTime - startTime;

metricLogger.info(String.format("%s %s %s %s", Operation.INSERT, msDto.getIntfcMsgObjId(), timeSpentInDb, timeSpentInCache, (timeSpentInDb - timeSpentInCache)));
```
Durable Memory
Results

<table>
<thead>
<tr>
<th>Tier</th>
<th>Transactions</th>
<th>Dependent Tx</th>
<th>Total Processing Time</th>
<th>CRUD Operation Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>50,000</td>
<td>100</td>
<td>73 minutes</td>
<td>42 minutes</td>
</tr>
<tr>
<td></td>
<td>50,000</td>
<td>100</td>
<td>44 minutes</td>
<td>3.5 minutes</td>
</tr>
</tbody>
</table>

THANK YOU!
References


