Achieving more with less – a real-world case study: Accelerating Quality Batch Delivery with Spring Batch

Feb 25, 2014

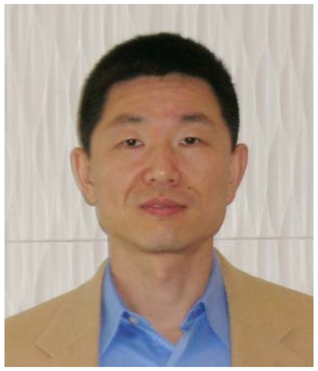
Yulin Zhao

Daugherty BUSINESS SOLUTIONS

EXPERIENCE BEYOND.

About the Speaker

- Yulin Zhao



- Manager, Application Architect at Daugherty Business Solutions.
- Over 14 years of consulting experience focusing on delivery management and technology architecture
- Delivered multi-million $ projects and batch, online and data center solutions for Fortune 100/500 companies and government agencies

Hobbies: swimming, finishing/hunting, real estate investment

Agenda

- Case Introduction (7 mins)
    - Background and Problem Statement
    - High-level Requirement
- Case Analysis (8 mins)
- The Spring Batch Solution (50 mins)
    - Sample Batch Job Design (5)
    - Sample Implementation (35)
        - Maximized leverage of out-of-box features
    - Best practices (10)
- Business Benefits Summary (5mins)
- Q & A (5 mins)
- Appendix

Accelerating Quality Batch Delivery with Spring Batch

CASE INTRODUCTION

Disclaimer

Contents of this case study and presentation are created from scratch based on experience with real companies. No private or proprietary information is contained.

Large Data Processing Company

Background:
Large Data Processing Company is a major consumer information solutions company and provides services to both individuals and businesses through integrating with various internal and external enterprise systems.

Problem Statement:
Individual Services Department (ISD) has 100s of batch jobs built in-house for data processing needs of all personal products, which account for a major part of the company's revenue. However, over the years, issues arising out of the batch jobs have significantly impacted customer satisfaction and ISD's ability to grow. Below are sample feedback from their customer support and operations departments:

- New customer registrations are down. What happened?
- There were errors 10 days ago. Why didn't we know?
- 60% Print products have issues reaching customers.
- "Batch cancellation process doesn't work."
- "I had to stay up all night and manually stop all other jobs on the server for X job to run …"

ISD is looking for a solution to address all these issues.

High Level Requirements

ISD decided to modernize its batch systems to support its strategic growth and to improve customer satisfaction. The solution should be a new batch architecture based on which jobs can be quickly developed, easily maintained yet still possessing the following qualities:

- Easily integrates with ISD’s internal and external systems
  - Primarily file based
- High fault tolerance
- Nearly guaranteed data integrity
- Easily scalable both vertically and horizontally
- Offers excellent operability:
  - Central administration
  - Custom notification for job events
  - Flexible auditing and logging per business logic
- Java-based, to better leverage existing resources’ skillset
- Solutions and components highly reusable
- Cost-effective

Accelerating Quality Batch Delivery with Spring Batch

CASE ANALYSIS

Case Analysis

- The requirements comprise primarily three aspects:

  - **Build a new batch architecture**
    - A batch framework should be created, selected or purchased to address common architectural concerns, while allowing jobs (business logic) to be developed and hooked in.

  - **Port existing jobs to / create new jobs on the architecture**
    - Existing jobs should be categorized and consolidated using the new batch framework
    - New jobs should be created with multi-tenant / multi-country and reusability in mind

  - **Create operations toolset**
    - Logging, auditing and notification
    - Central job administration tool

Architectural Issues and Challenges

ISD’s pains, in summary, are caused by issues and challenges in the following areas of their in-house batch architecture:

- **Reliability**
  - No overall design for ensuring data consistency
  - Lack of a systematic exception handling approach across systems
- **Robustness**
  - Fault tolerance is low, data formatting issues causes job to abend
- **Performance**
  - Buggy multi-threading solution
  - Major change needed to scale horizontally
- **Operability**
  - No usable exception logging or audit trail
  - No notifications for fatal errors
  - Data integrity issues cause constant manual intervention
- **Maintainability**
  - System enhancement has been very challenging causing longer time to market

Solution Approach Comparison*

Considerations	Revamp Existing Systems	Rebuild with Spring Batch	Rebuild with IBM WebSphere Compute Grid
Achieve Required Architectural Qualities	Very difficult due to lack of design knowledge	YES	YES
Central Admin	Yes, but doesn't always work	Yes, need customization	YES
Batch Container	In-house framework	Self-provided	WAS provided
Need to run within App Server	Some do; some don't	No, but can.	YES
Integration with transaction, security and other infrastructure services	Partial	Light-weight, highly customizable	Fully Integrated, heavy weight
Product Cost	N/A	Free	$$$$$$

* Not exhaustive listing of all comparisons by ISD.

Spring Batch Features Quick Review

➢ Spring Batch (SB), as a leading open-source Java batch framework, has deliberate design and features to tackle common challenges ISD is facing:

- Reliability
  - Offers persistent mechanism to track job execution
    - Business agnostic statistics collected for monitoring & analysis
    - Job restart enabled to resume execution from where it left off
  - Chunk and step level transaction management
- Robustness
  - Declarative skip processing
    - Triggered by pre-configured exceptions
    - Can be logged / audited by skip listeners
    - Controlled by skip limit or policy
  - Declarative retry processing
    - Triggered by pre-configured exceptions
    - Can be logged / audited by retry listeners, if necessary
    - Controlled by retry limit or policy
  - Errors affect only the current chunk

Spring Batch Features Quick Review

- Spring Batch (SB) has deliberate design and features to tackle common challenges ISD is facing:

  - Performance
    - Chunk oriented processing and configurable chunk size
    - Local and Remote Partitioning
    - Parallel /multi-threaded Steps
    - Remote chunking

  - Operability
    - Provides listener hooks by configuration at various levels of job life-cycle events that allow customized auditing, logging and notification
    - Spring Batch Admin is provided out of box

  - Integration-friendly : works seamlessly with Spring Integration

  - Maintainability
    - SB is a well-documented, widely-used open-source Java batch framework.

Quiz

- Spring Batch is a great framework, but why are most of its readers and writers not thread-safe?

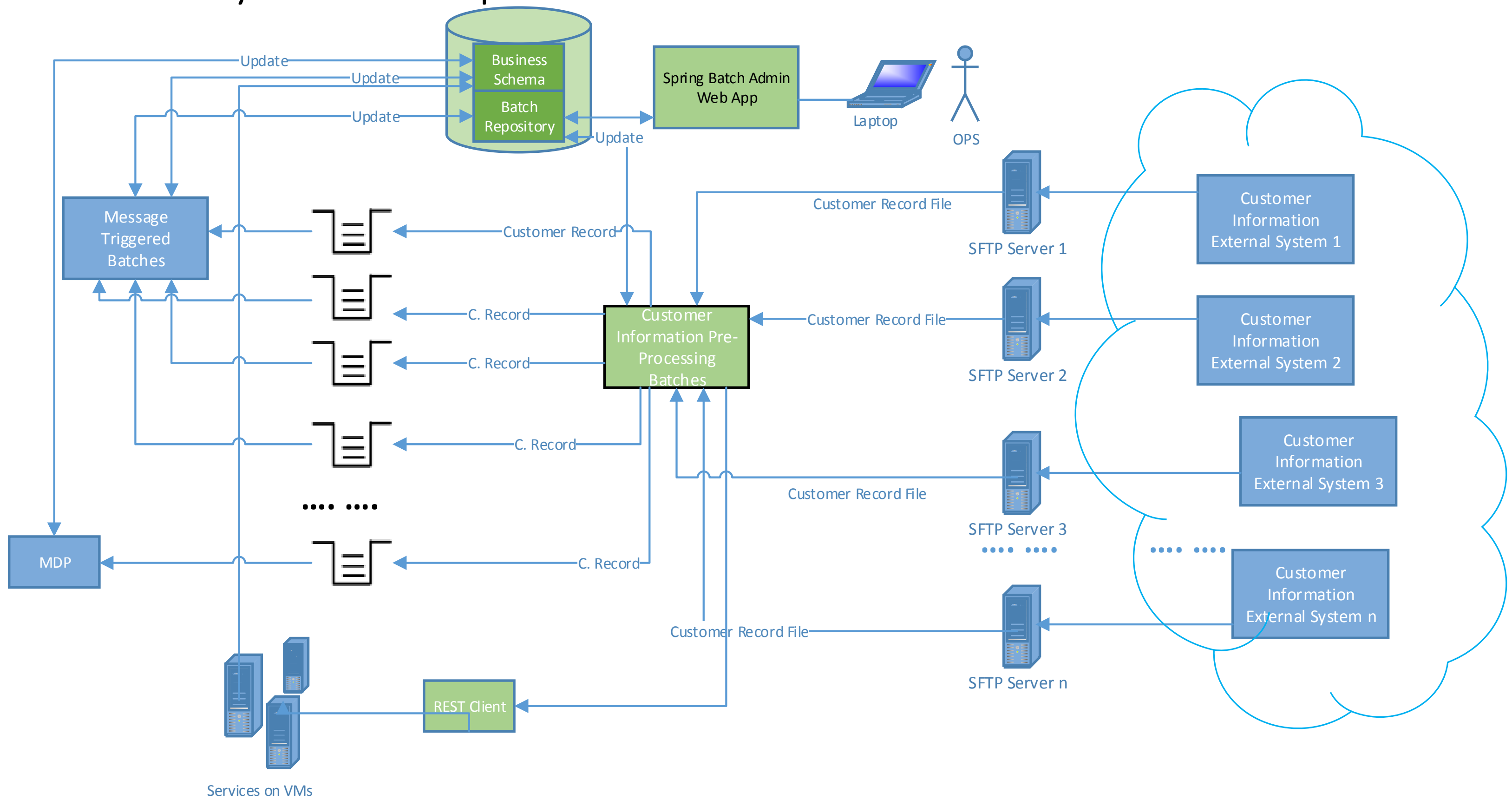
Accelerating Quality Batch Delivery with Spring Batch

THE SPRING BATCH SOLUTION

The Spring Batch Approach

Sample Batch Job Design (partial):

- A multi-tenant solution for customer information monitoring.
    - Flow: daily information update and alert.



Daugherty BUSINESS SOLUTIONS

EXPERIENCE BEYOND

The Spring Batch Approach

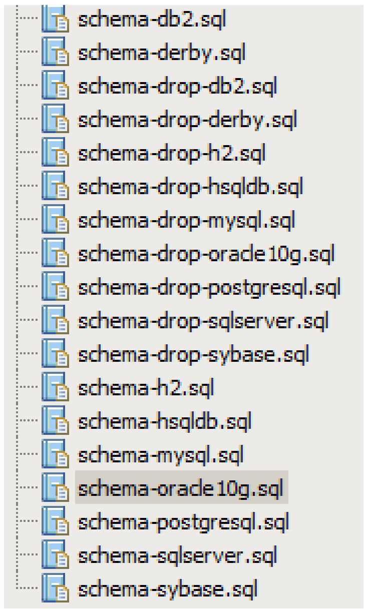
Sample Batch Job Implementation:

Reliability: leveraged out-of-box SB repository for job execution tracking and restartability.

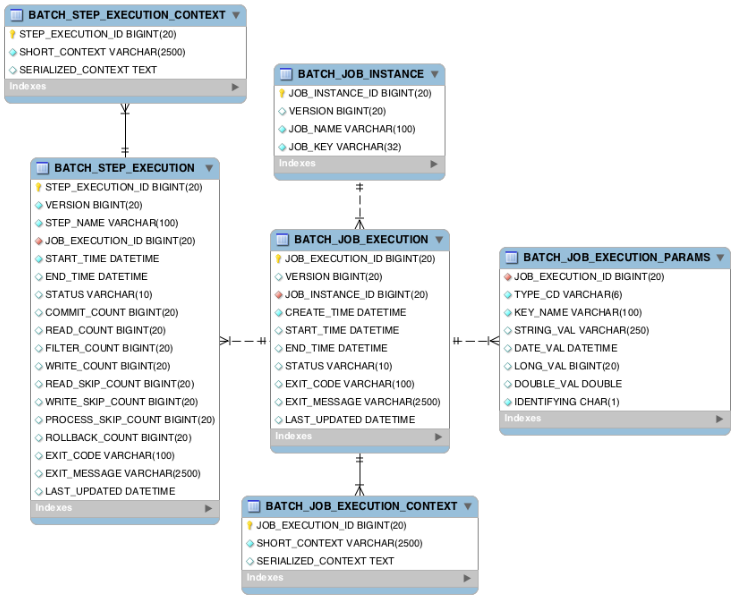
➢ Used job repository DDL provided by spring-batch-core-2.x.x.RELEASE.jar under org.springframework.batch.core package.

Considered:

- Double VARCHAR column size and <JobRepository…max-varchar-length> for longer exceptions or multi-byte char sets
- Change table prefix in in DDL <JobRepository>



```
schema-db2.sql
schema-derby.sql
schema-drop-db2.sql
schema-drop-derby.sql
schema-drop-h2.sql
schema-drop-hsqldb.sql
schema-drop-mysql.sql
schema-drop-oracle10g.sql
schema-drop-postgresql.sql
schema-drop-sqlserver.sql
schema-drop-sybase.sql
schema-h2.sql
schema-hsqldb.sql
schema-mysql.sql
schema-oracle10g.sql
schema-postgresql.sql
schema-sqlserver.sql
schema-sybase.sql
```



Daugherty
BUSINESS SOLUTIONS

EXPERIENCE BEYOND

The Spring Batch Approach

Sample Batch Job Implementation:

Operability: listener call-backs allow for customized audit/logging/notification.

Robustness: expected issues with individual records can be tolerated.

- Listeners

```
<job id="process-data-nfs-db" xmlns="… ">
      <listeners>
            <listener ref="jobListener " />                                    ➔Job Level
      </listeners>
      <step id="ftp " next="parseDataFile">
            <tasklet ref="getDataFile" transaction-manager="batchTransactionManager" />
                  <listeners>
                        <listener ref="propertiesLoader " />                   ➔Step Level
                  </listeners>
      </step>
      <step id=" parseDataFile" next="….. ">
            <tasklet transaction-manager="batchTransactionManager">
                  <chunk reader="dataReader " processor="dataProcessor "
                        writer="dataWriter " skip-limit="20" commit-interval="200">
                        <skippable-exception-classes>
                              <include
                              class="org.springframework.batch.item.file.FlatFileParseException" />
                        </skippable-exception-classes>
                  </chunk>
                  <listeners>
                        <listener ref="propertiesLoader" />
                        <listener ref="skipListener" />                        ➔Action when skip occurs
                  </listeners>
            </tasklet>
      </step>
….
</job>
```

- ✓ **Listeners can be at many levels and during different events**
- ✓ **Use "REQUIRES_NEW" propagation level to when logging in DB.**

Daugherty
BUSINESS SOLUTIONS

EXPERIENCE BEYOND

The Spring Batch Approach

Sample Batch Job Implementation:

Listeners are developed separately focusing on business logic and plugged in.

➢ Listeners

- Job listener

```
<!-- this listener notifies OPS team if Job doesn't complete successfully -->
<bean id="jobListener" class="com…….batch.listener.PostJobListener">
      <property name="notifier" ref="emailNotifier"/>
      <property name="templateMessage" ref="templateMessage"/>
</bean>
```

- Skip listener

```
<!-- this listener simpliy logs the skipped record -->
<bean id="skipListener" class="com.....batch.listener.FileParseSkipListener" />
```

- Properties loader listener (step execution listern – before step)

```
<!-- this listener loads corresponding properties file per tenant parameter-->
<bean id="propertiesLoader" class="com.....batch.utility.PropertyLoader" scope="step">
      <property name="tenantProperties">
            <bean class="org.springframework.beans.factory.config.PropertiesFactoryBean">
                  <property name="location”
                  value="classpath:properties/batch.#{jobParameters['tenant']}.properties"
                  />
            </bean>
      </property>
</bean>
```

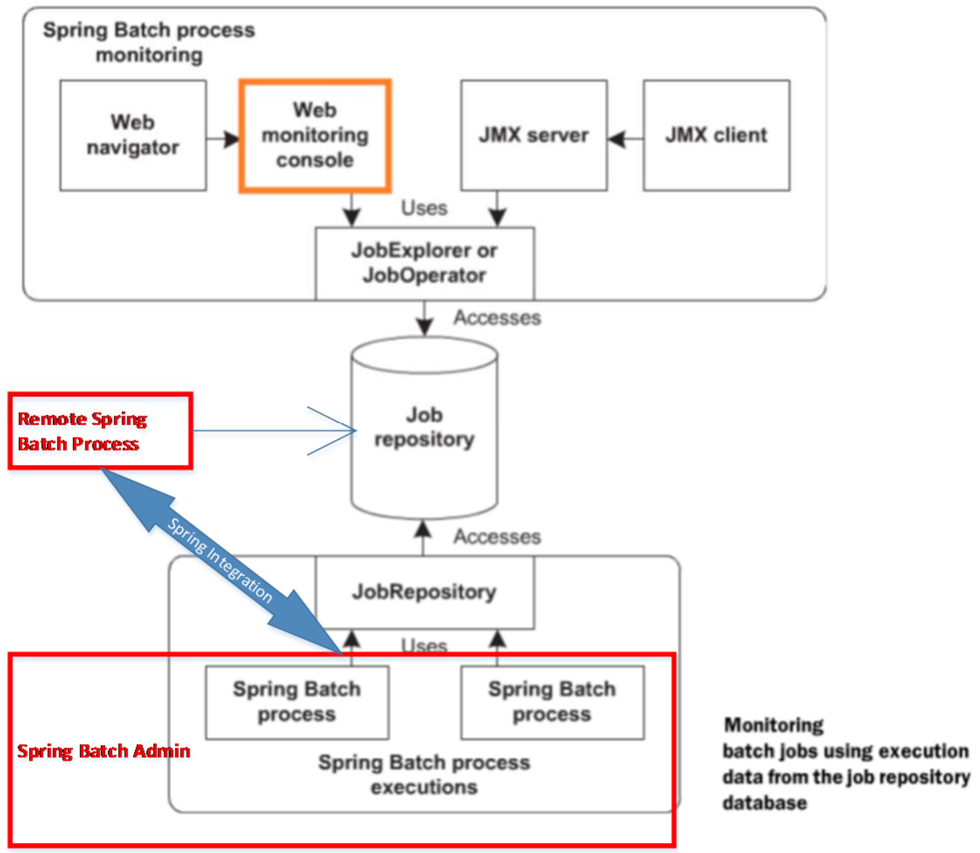
✓ **JobExecution, Item/Exception, StepExecution are passed to the above listeners for business logic.**

✓ **Listeners can be used for other purposes such as lazy-loading a properties file**

The Spring Batch Approach

Sample Batch Job Implementation:

Operability: transaction-level monitoring made possible by SB repository and monitoring architecture.


Spring Batch process monitoring
Web navigator
Web monitoring console
JMX server
JMX client
Uses
JobExplorer or JobOperator
Accesses
Job repository
Remote Spring Batch Process
Spring Integration
Accesses
JobRepository
Uses
Spring Batch process
Spring Batch process
Spring Batch Admin
Spring Batch process executions
Monitoring batch jobs using execution data from the job repository database

- **Spring Batch Admin (SBA)**
  - **Offers visibility to all jobs sharing the same repository**
  - **By default only allows starting of local jobs within the same server container using multi-threads.**
  - **Can be extended to start remote jobs via messaging/Spring Batch Integration**
- **Consider: Spring XD**
  - **Has SBA functionality with distributed model built-in**
  - **Uses much of SBA under the hood**

Daugherty BUSINESS SOLUTIONS

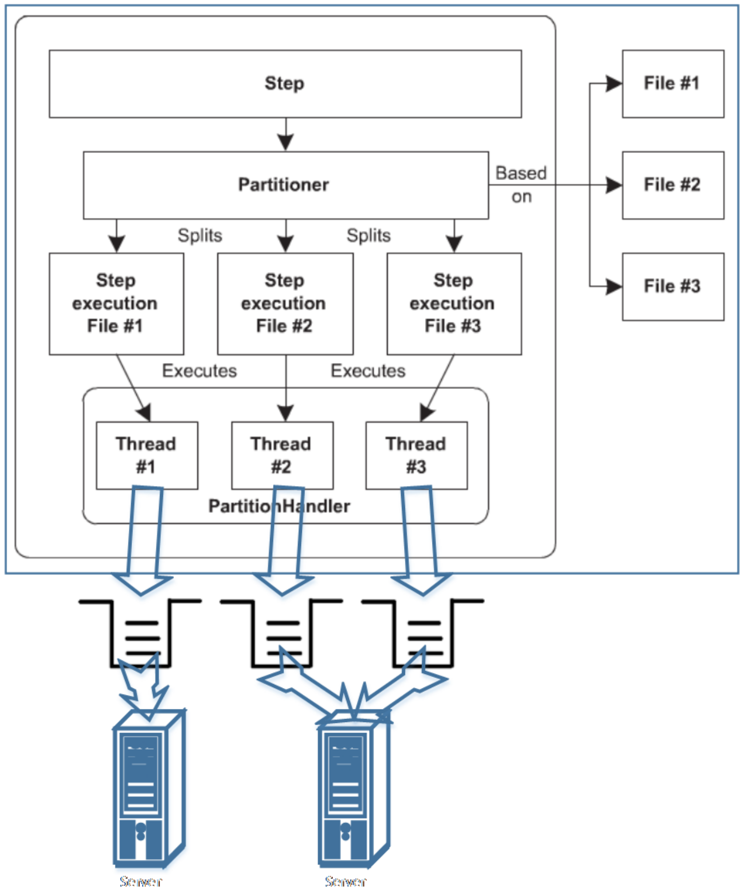
EXPERIENCE BEYOND

The Spring Batch Approach

Sample Batch Job Implementation:

Performance: leveraged partitioning for parallel input parsing locally and JMS queues for distributing processing load to multiple nodes

- **Step 1: input split (not shown)**
- **Step 2: custom partitioner delegates to resource partitioner for file partition**
- **Step 3: custom partitioner assigns JMS Queues to partitions by storing queue names to the execution context for each partition**
- **Step 4: JMS queues are configured as step scope bean. Late binding is used to get queue names from step execution context**
- **Step 5: Processing nodes use Message triggered batch, scheduled batch or Message Driven POJO to process load**



The Spring Batch Approach

Sample Batch Job Implementation:

Performance: leveraged partitioning for parallel input parsing locally; and JMS queues for distributing processing load to multiple nodes

```
<batch:job id="alertJob" xmlns="...">
       <batch:step id="loadBalancingStep">
              <batch:partition step="partitionLoadBalancingStep“ partitioner="queueLoadBalancingPartitioner">
                     <batch:handler grid-size=“10" task-executor="taskExecutor" />
              </batch:partition>
       </batch:step>
</batch:job>
<batch:step id="partitionLoadBalancingStep">
       <batch:tasklet>
              <batch:chunk reader="inputXMLReader" processor="alertProcessor"
              writer="loadBalancingWriter" commit-interval=“150" />
       </batch:tasklet>
</batch:step>

<!- allocates partitions created by resource partitioner to JMS queues for  load balancing-->
<bean id="queueLoadBalancingPartitioner" scope="step“ class="com.....batch.gridding.JMSQueueLoadBalancingPartitioner">
       <property name="resourcePartitioner" ref="filePartitioner" />                  ➔ delegating to SB
       <property name="tenantName" value="#{jobParameters['tenantName']}" />
       <property name="queueBaseName" value="#{jobParameters[‘queueBaseName']}" />
</bean>
<bean id="filePartitioner"
       class="org.springframework.batch.core.partition.support.MultiResourcePartitioner">
       <property name="keyName" value="fileName" />
       <property name="resources“ value="file:/... /gridding/input/*.xml" />
</bean>
… …
<!--  Late binding of the queue names generated by partitioner     -->
<bean id="loadBalancingQueue" class="org.apache.activemq.command.ActiveMQQueue" scope="step">  ➔ same bean name shared
    <constructor-arg value="#{stepExecutionContext[queueName]}" />                                 by multiple parallel
</bean>                                                                                              steps
```

The Spring Batch Approach

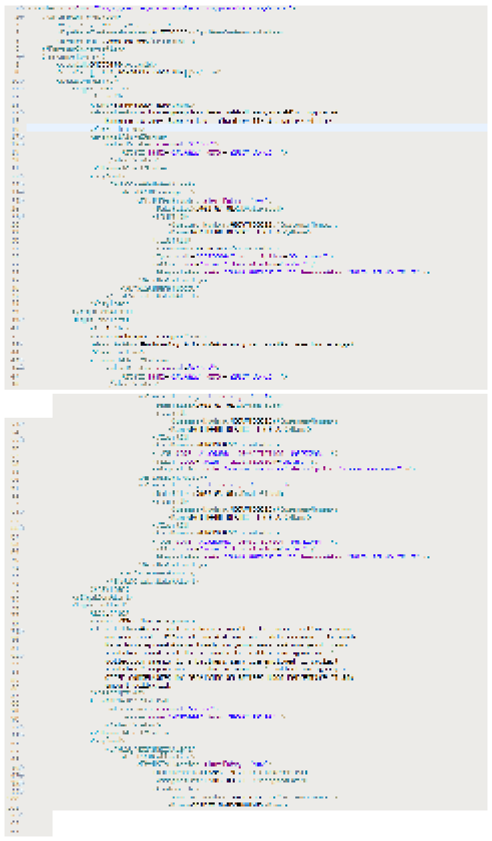
<u>Sample Batch Job Implementation:</u>

Used SB “Power Readers” to accelerate delivery.

➢ Readers - XML

<u>Configuration:</u>

```
<bean id="inputXMLReader"  scope="step"
            class="org.springframework.batch.item.xml.StaxEventItemReader" >
     <property name="fragmentRootElementName" value="Record" />
     <property name="resource"  value="#{stepExecutionContext[fileName]}"/>
     <property name="unmarshaller" ref=“idAlertUnmarshaller" />
</bean>
<bean id="alertDataReader"  scope="prototype"
            class="org.springframework.batch.item.xml.StaxEventItemReader">
     <property name="fragmentRootElementName" value="record" />
     <property name="unmarshaller" ref="alertDataUnmarshaller" />
</bean>
```


[illegible]

- ✓ **Stax-based high-performance XML Reader**
- ✓ **Only about a dozen lines of code written to trigger further parsing XML embedded in CDATA section**
- ✓ **Spring OXM for mapping XML to domain objects**
- ✓ **Job Restart enabled automatically**

Daugherty BUSINESS SOLUTIONS

EXPERIENCE BEYOND

The Spring Batch Approach

Sample Batch Job Implementation:

Used SB “Power Readers” to accelerate delivery.

- Readers – XML: code processing xml within CDATA section

```
/**
 * Processes all records for a registered customer.
 */
public DataRecord process(DataRecord records) throws Exception {

        // get record data from the CDATA section
        String recordData = records.getRecordData();

        // Converts embedded xml into a feeding data source
        ByteArrayResource recordDataResource = new ByteArrayResource(recordData.getBytes());

        alertDataReader.setResource(recordDataResource);
        try {
                alertDataReader.open(new ExecutionContext());       //➔new context for inner parsing
                 AlertDataRecord record = alertDataReader.read();   //➔inner parsing
                while (record != null) {
                        // build record list
                        records.getRecordList().add(record);
                        record = alertDataReader.read();
                }
        } finally {
                alertDataReader.close();
        }
        return records;
}
```

Daugherty BUSINESS SOLUTIONS

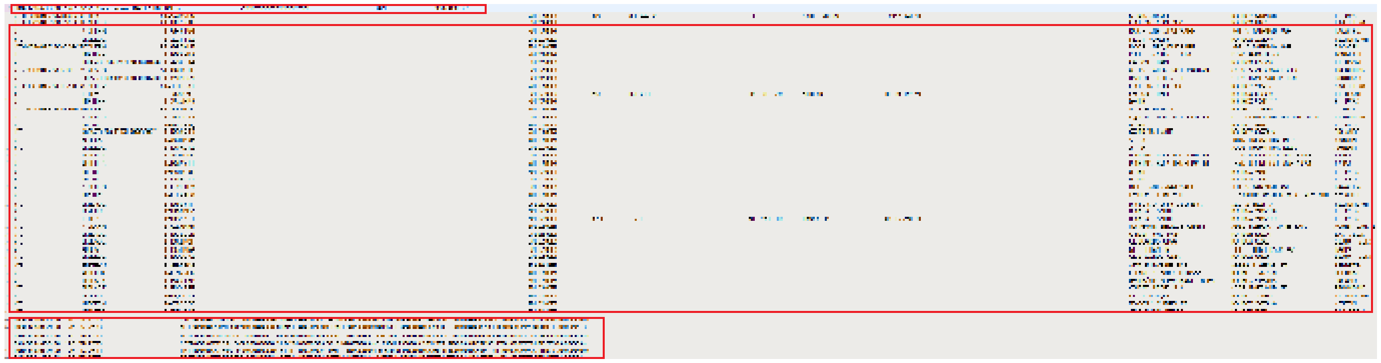
EXPERIENCE BEYOND

The Spring Batch Approach

<u>Sample Batch Job Implementation:</u>

Used SB “Power Readers” to accelerate delivery.

- Readers – Flat File



```
<bean id="dataFileReader" class="org.springframework.batch.item.file.FlatFileItemReader" scope="step">
  <property name="resource" value="file:#{jobExecutionContext.get('batchProperties').
  getProperty('....response.localDirectory')}#{jobExecutionContext['fileName']}"/>
        <property name="lineMapper">
                <bean class="org.springframework.batch.item.file.mapping.DefaultLineMapper">
                        <property name="lineTokenizer" ref="dataRecordLinePatterns" />
                        <property name="fieldSetMapper" ref="dataRecordMapper"/>
                </bean>
        </property>
</bean>
```

The Spring Batch Approach

Sample Batch Job Implementation:

Used SB “Power Readers” to accelerate delivery.

➢ Readers – Flat File

```
<bean id= "dataRecordMapper" class="com.......batch.mapping.DataRecordMapper" />
<bean id="dataRecordLinePatterns" class="org.springframework.batch.item.file.transform.PatternMatchingCompositeLineTokenizer">
                    <property name="tokenizers">
                              <map>
                                        <entry key="HEADER*" value-ref="headerTokenizer" />
                                        <entry key="RECORD*" value-ref="recordTokenizer" />
                                        <entry key="TRAILER*" value-ref="trailerTokenizer" />
                              </map>
                    </property>
</bean>
<bean id="headerTokenizer“ class="org.springframework.batch.item.file.transform.FixedLengthTokenizer">
          <property name="columns" value="1-3,13-20,67-106,199-200" />
          <property name="names" value=" lineType, date, customerName, headerEnd" />
</bean>
<bean id="recordTokenizer“ class="org.springframework.batch.item.file.transform.FixedLengthTokenizer">
          <property name="columns" value="1-4, 5-21, 22-26, 27-28, 29-36, 37-44, ......., 999-1000" />
          <property name="names" value="lineType, id, displayCode, businessType, inquiryDate, amount, ......, accountCode" />
</bean>
<bean id="trailerTokenizer“ class="org.springframework.batch.item.file.transform.FixedLengthTokenizer">
          <property name="columns" value="1-3,4-10,299-300" />
          <property name="names" value=" lineType, companyID, TriggerTrailerEnd" />
</bean>
```

- ✓ **Only code written is the mapper**
- ✓ **Pattern: regex-like (supports ‘*’ and ‘?’)**
- ✓ **Must provides end segment mapping even it’s not used**

The Spring Batch Approach

Best Practices:

- **Performance**
  - Keep Meta Data Store in the same business data schema if possible; otherwise use Database Synonyms to avoid true global transactions.
  - Use chunk-oriented processing whenever possible.
  - Use local step partitioning or parallel steps over multi-threaded steps
    - Most of SB out-of box readers/writers are not thread-safe
- **Maintainability**
  - Separate job, step, common and infrastructure configuration files in different folders.
  - Separate / Delegate business logic to non-batch components.
  - Use parent jobs/steps for common scenarios handling such as notification
  - Use simplest scaling technique that works.
  - Minimize coupling between steps when designing them.
  - Create and use a common job naming convention such as <action>-<from>-<to> for integration oriented jobs.

The Spring Batch Approach

Best Practices:

- **Coding**
  - Do not reinvent the wheels – spend time understanding Spring Batch OOB features.
  - Use Spring EL and late binding to access execution context and Job Parameters
  - Design job parameters for readers and writers from the start for restartability.
    - E.g.: same dataset needs be ensured for later runs for counter-based readers/writers to restart correctly
- **Operability**
  - Use listeners to monitor important events and notify operations as needed.
  - Thoroughly test restart on staging or pre-production environment.

Common Pitfalls

- Forget "step" scope when` using late binding to access step execution context
- Not aware of implicit SB naming convention and refactoring causes issues.
- Think a Spring Batch job can always automatically restart from where it left off, not knowing the readers and writers must keep track of the "where" and store it in the execution context.
- Sharing data with Spring beans for step communication without clearing them after each execution when run under a web container

Accelerating Quality Batch Delivery with Spring Batch

BUSINESS BENEFITS SUMMARY

Business Benefits

Executive Summary

- ISD has 300+ batch jobs supporting various products. The Spring-Batch based solution has started to show significant benefits through the 1st two projects.
  - Reduced average batch job development effort by at least 50%*
  - Shortened 1 batch job's run window by at least 70%
  - Enabled Rapid Multi-Region / Multi-country roll-out
  - Reduced Compliance risk with transaction tracking / auditing
  - Expect to reduce # of batch jobs by ~80% due to multi-tenant designs
  - Reduction of code maintenance effort is expected to exceed 80%
  - Expect to reduce Operations Support by at least 60%

* Learning curve accounted for.

Business Benefits

Business Objective	Before	Now / Expected
Reduce development and maintenance effort	• <u>Customer Information Parser 1</u>:~1200 lines of code	• 62 lines of code + 43 lines of configuration – using SB's standard XML readers
	• <u>Customer Information Parser 2</u>: ~5400 lines of code (not counting domain/utilities code)	• ~300 lines of code + ~500 lines of XML – using SB's flat file, fixed length & pattern matching readers
	• Number of Jobs needed: 300+	• Estimated to be ~ 50 or less – with multi-tenant, multi-country support. Built reusable jobs with standard SB components for multiple scenarios. Such as FTP, unzipping etc.

Business Benefits

Business Objective	Before	Now / Expected
Increase Customer Satisfaction	• Customer or partner is the "first line of defense" detecting issues • When issues occurred, jobs aborted leaving inconsistent data records. Manual intervention is needed to remedy data integrity. • Operations team's routine job became manually manipulating database to keep the business running.	• Operations team and customer support are notified as soon as issues occurred. Audit trail and logging are readily available for quickly pinpointing issues. After issues are resolved, jobs are restarted from where they left off. • Data Integrity issue significantly reduced due to SB's transaction management at the chunk level.
Rapid Multi-region roll-out	• Not supported. A set of jobs need be created for each country. The total code to maintain = (lines of code / country) x (# of countries)	• Supported. Components are developed to capture differences from country to country and are plugged into the same job when running.

Business Benefits

	Before	Now / Expected
Reduce Compliance Risk	• High risk of being sanctioned by Consumer Financial Protection Bureau • Facing litigations	• Low compliance risk – due to audit/logging in place.
Reduce business impact due to performance issues	• Consumer Information Parser Job1 run 1+ hours for average production load • Consumer Information Processing Job2 took up to 10-20 hrs. having had severe business impact	• Shortened Job1 run window by 70% using lower hardware and peak production load. • Expect to contain worse-case run window within 5 hrs. by using SB's built-in caching, vertical and horizontal scaling mechanisms.

Accelerating Quality Batch Delivery with Spring Batch

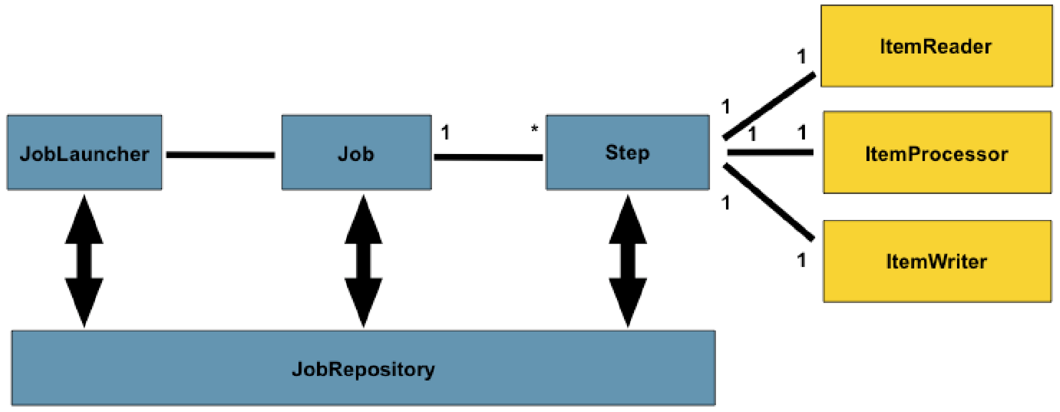
Q & A

Accelerating Quality Batch Delivery with Spring Batch

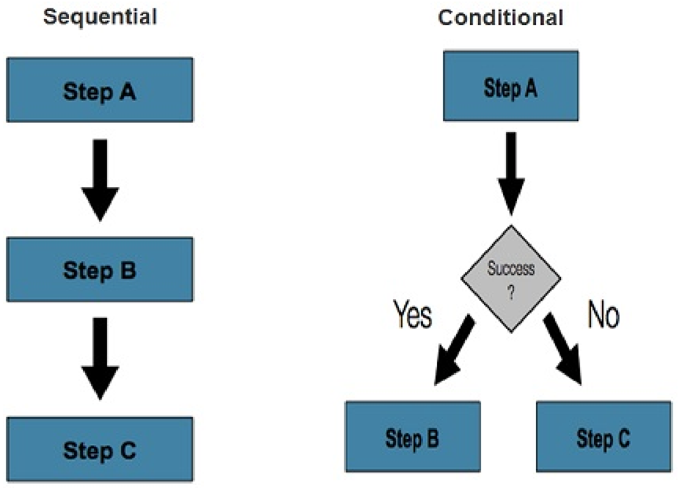
APPENDIX

Spring Batch Architecture - stereotype

- Framework Stereotype

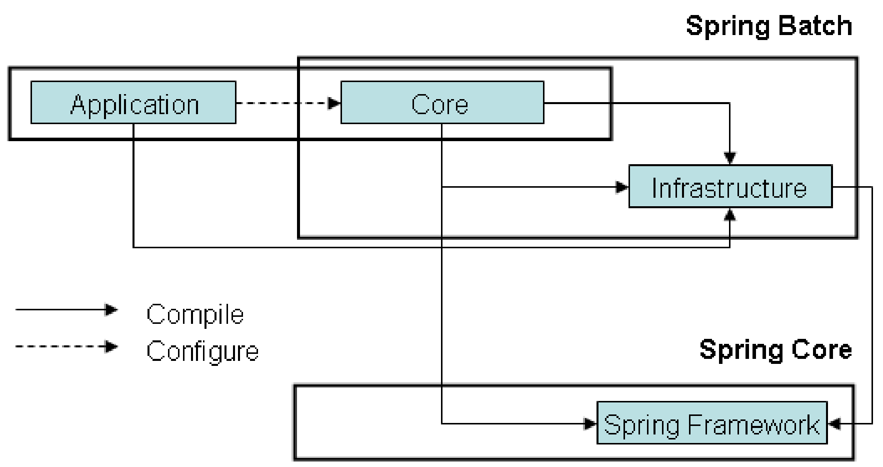


- Step Stereotype

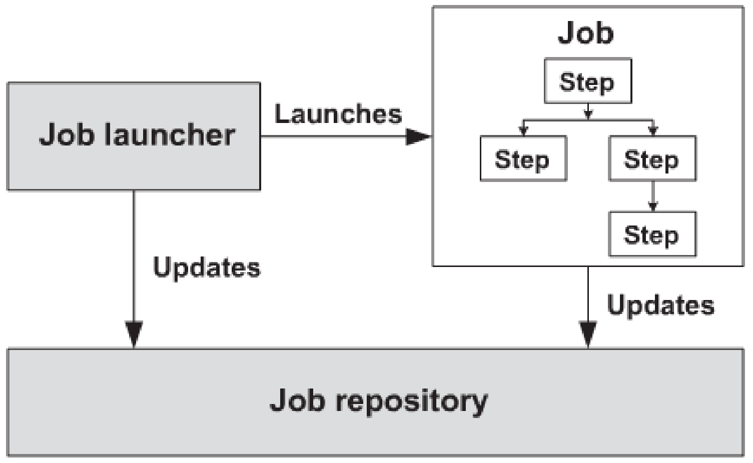


Spring Batch Architecture – Major Components

- Major Components – implementation view



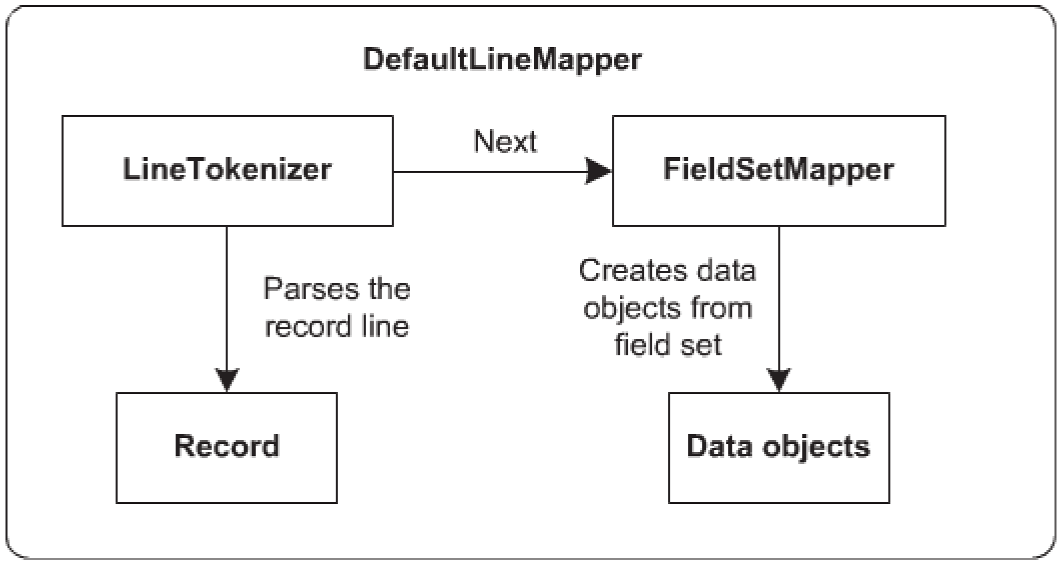
- Major Components – logical view



Daugherty BUSINESS SOLUTIONS

EXPERIENCE BEYOND

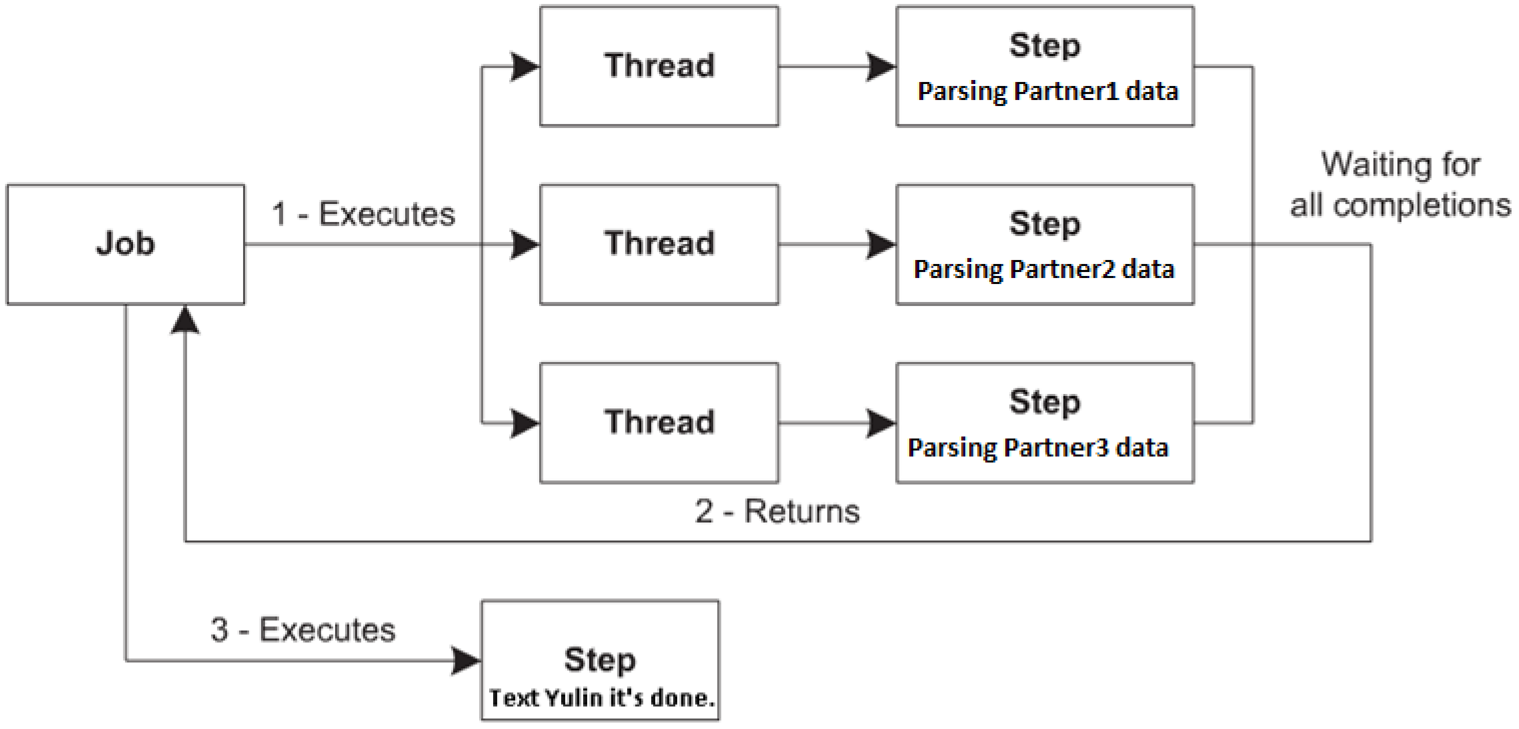
Spring Batch Utility – DefaultLineMapper



Interactions between `LineTokenizer` and `FieldSetMapper` in a `DefaultLineMapper`. The `LineTokenizer` parses record lines, and the `FieldSetMapper` creates objects from field sets.

Spring Batch – Parallel Steps

Parallel Steps run by multi-threads with each thread using chunk oriented processing (cached reading/processing of multi-records for 1 transaction).



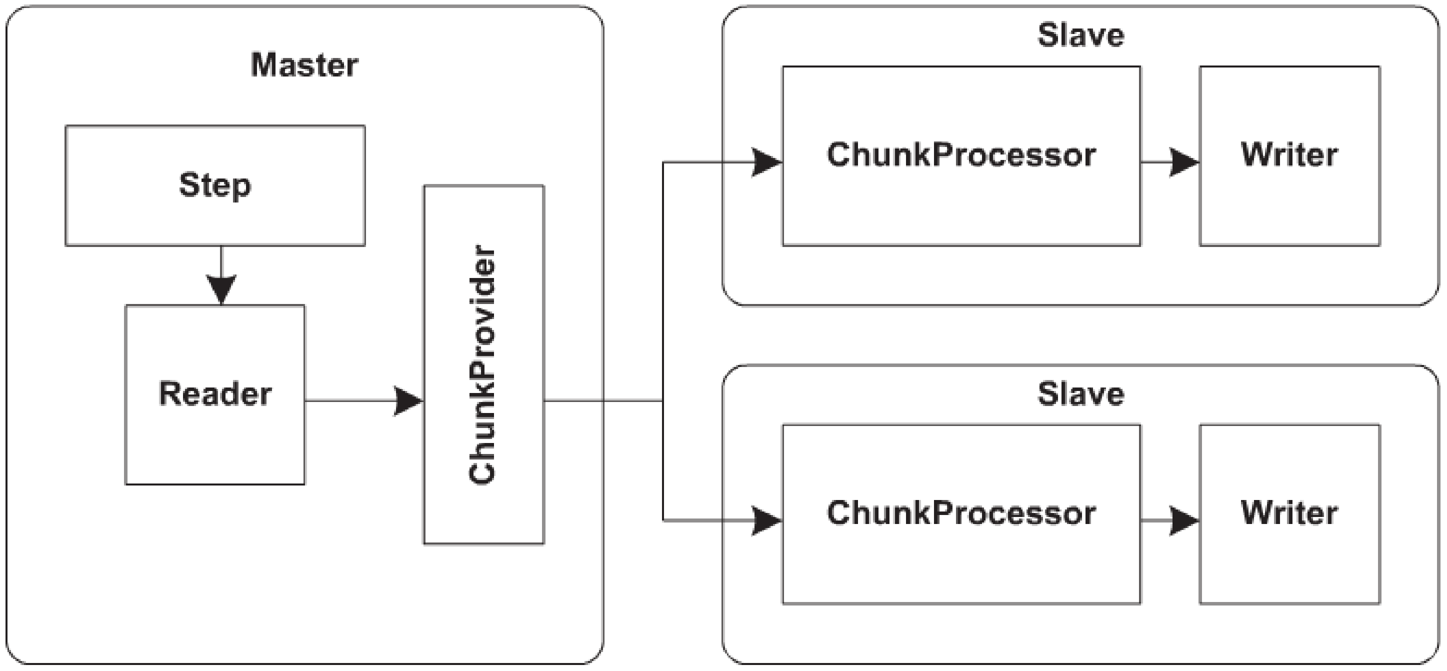
Executing steps in parallel using dedicated threads

Daugherty BUSINESS SOLUTIONS

EXPERIENCE BEYOND

Spring Batch – Remote Chunking

Workload distributed to multiple machines



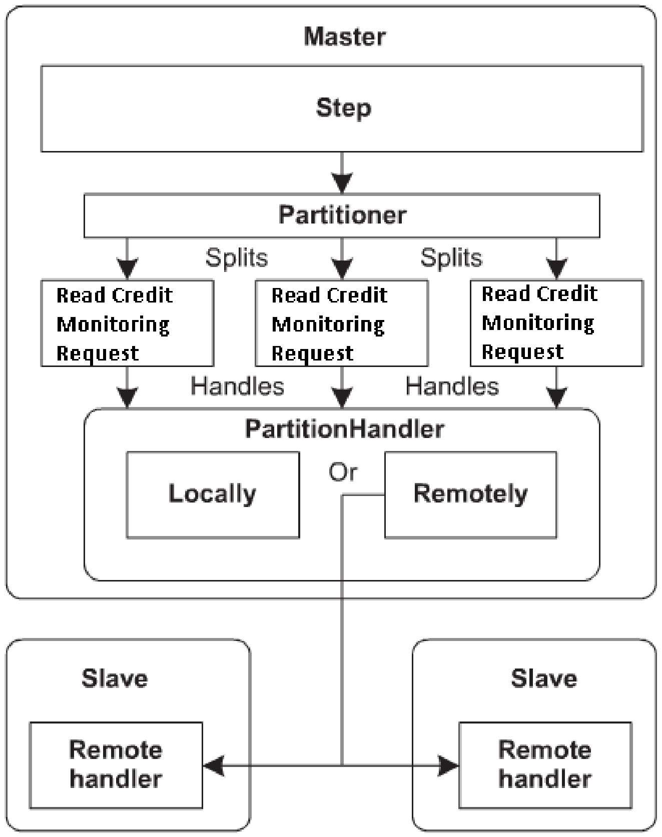
Remote chunking with a master machine reading and dispatching data to slave machines for processing

Daugherty BUSINESS SOLUTIONS

EXPERIENCE BEYOND

Spring Batch – Partitioning

Partition input then dispatch input to local handler (mult-threads) or remote handler (multi-nodes) for processing



Partitioning splits input data processing into several step executions processed on the master or remote slave machines.

Daugherty BUSINESS SOLUTIONS

EXPERIENCE BEYOND

References and contact

Web site

- spring.io

Books

- Cogoluegne (2012). *Spring Batch in Action.* Shelter Island, NY: Manning Publications.
- Minella (2012). *Pro Spring Batch.* New York: Apress.

Contact:

- Email: Yulin.Zhao@Daugherty.com
- Skype: YulinZhao@msn.com