Aprendizaje Automático sobre Grandes Volúmenes de Datos

Clase 13

Pablo Ariel Duboue, PhD

Universidad Nacional de Córdoba,
Facultad de Matemática, Astronomía y Física

Material de lectura

- Clase pasada:
  - Scalable Scientific Computing Algorithms Using MapReduce por Xiang Jingen, Master of Mathematics UWaterloo '13
    - https://uwspace.uwaterloo.ca/bitstream/handle/10012/7830/Xiang_Jingen.pdf
  - Parallelized stochastic gradient descent por Zinkevich, Weimer, Li, Smola (NIPS 2010)
- Ésta clase:
  - Alternating Direction Method of Multipliers (Boyd et al. 2011)
    - web.stanford.edu/~boyd/papers/pdf/admm_distr_stats.pdf
  - MPI (http://en.wikipedia.org/wiki/Message_Passing_Interface)
    - http://www.mpich.org/
  - Colas de pasaje de mensajes (http://en.wikipedia.org/wiki/Advanced_Message_Queuing_Protocol)
    - http://activemq.apache.org/

Preguntas

- ¡Sin preguntas!

Recordatorio

- El sitio Web de la materia es http://aprendizajengrande.net
  - Allí está el material del curso (filminas, audio)
- Leer la cuenta de Twitter https://twitter.com/aprendengrande es obligatorio antes de venir a clase
  - Allí encontrarán anuncios como cambios de aula, etc
  - No necesitan tener cuenta de Twitter para ver los anuncios, simplemente visiten la página
- Suscribirse a la lista de mail en aprendizajengrande@librelist.com es optativo
  - Si están suscriptos a la lista no necesitan ver Twitter
- Feedback para alumnos de posgrado es obligatorio y firmado, incluyan si son alumnos de grado, posgrado u oyentes
  - El "resúmen" de la clase puede ser tan sencillo como un listado del título de los temas tratados

Temas Claves

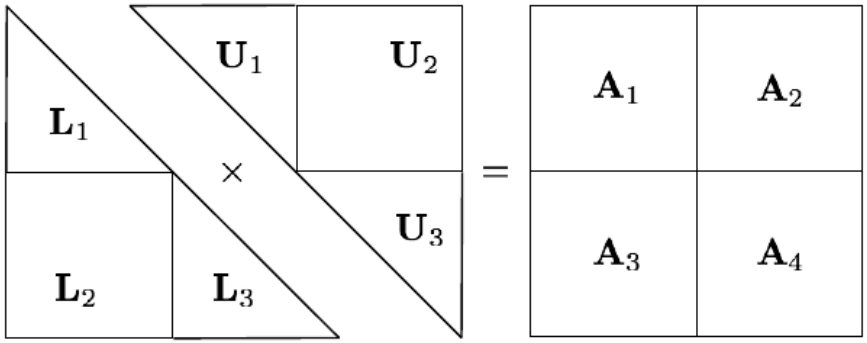
- Identificación de la máxima tarea que puede hacerse por nodo
- Descomposición de la tarea normal en tareas por nodo
- Tareas globales, realizadas en un nodo (central) y tareas paralelas
- Comunicación entre tareas mediante HDFS y archivos bandera
- Tareas que sólo hacen Map
- Cálculos intermedios

Descomposición LU

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ a_{31} & a_{32} & \dots & a_{3n} \\ \dots & & & \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} = \begin{pmatrix} l_{11} & & & \\ l_{12} & l_{22} & & \\ l_{13} & l_{23} & l_{33} & \\ \dots & & & \\ l_{n1} & l_{2n} & \dots & l_{nn} \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ & u_{22} & \dots & u_{2n} \\ & & u_{33} & \dots \\ & & & \dots \\ & & & u_{nn} \end{pmatrix}$$

(adaptado de Xiang Jingen, 2013)

Descomposición LU en MR



(adaptado de Xiang Jingen, 2013)

Descomposición LU en MR

$$
\begin{aligned}
\mathbf{L}_1\mathbf{U}_1 &= \mathbf{P}_1\mathbf{A}_1, \\
\mathbf{L}_1\mathbf{U}_2 &= \mathbf{P}_1\mathbf{A}_2, \\
\mathbf{L}_2^{'}\mathbf{U}_1 &= \mathbf{A}_3, \\
\mathbf{L}_3\mathbf{U}_3 &= \mathbf{P}_2(\mathbf{A}_4 - \mathbf{L}_2^{'}\mathbf{U}_2), \\
\mathbf{L}_2 &= \mathbf{P}_2\mathbf{L}_2^{'},
\end{aligned}
$$

(adaptado de Xiang Jingen, 2013)

El algoritmo a vuelo de pájaro

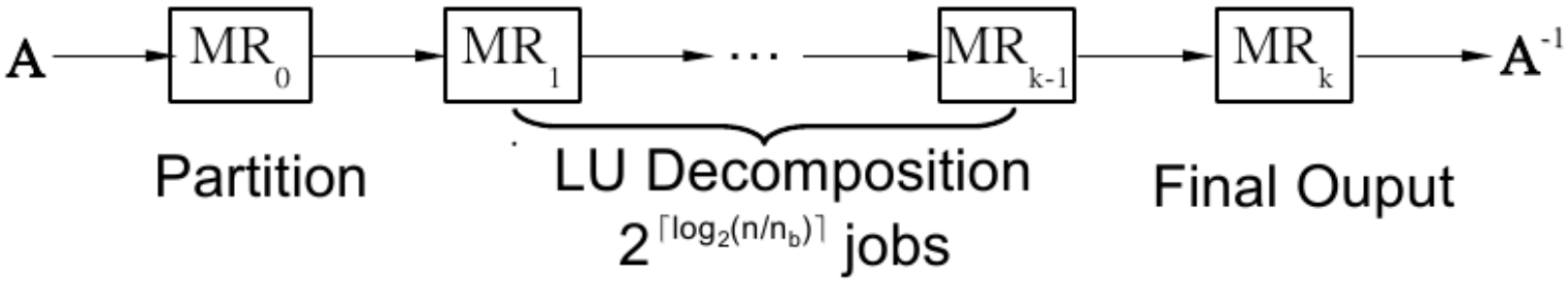
- Dividir la matriz original recursivamente hasta que $\mathbf{L}_1\mathbf{U}_1 = \mathbf{P}_1\mathbf{A}_1$ sea soluble en una máquina
- Descomponer $\mathbf{A}_1$
- Calcular $\mathbf{U}_2$
- Calcular la permutación de $\mathbf{L}_2$, $\mathbf{L}'_2$
- Calcular $\mathbf{B} = \mathbf{A}_4 - \mathbf{L}'_2\mathbf{U}_2$
- Descomponer $\mathbf{B}$
- Armar la salida total a partir las partes

Implementación en MapReduce

1. Nodo master crea archivos de control que son usados como entrada a las tareas map subsecuentes
   - Falsas entradas
2. Una tarea MapReduce subdivide $\mathbf{A}$ de manera recursiva
3. Se envian tareas MapReduce para $\mathbf{L}_2'$, $\mathbf{U}_2$ y $\mathbf{B}$
   - $\mathbf{L}_1$ y $\mathbf{U}_1$ son ejecutadas en el master node si $\mathbf{A}_1$es lo suficientemente pequeña

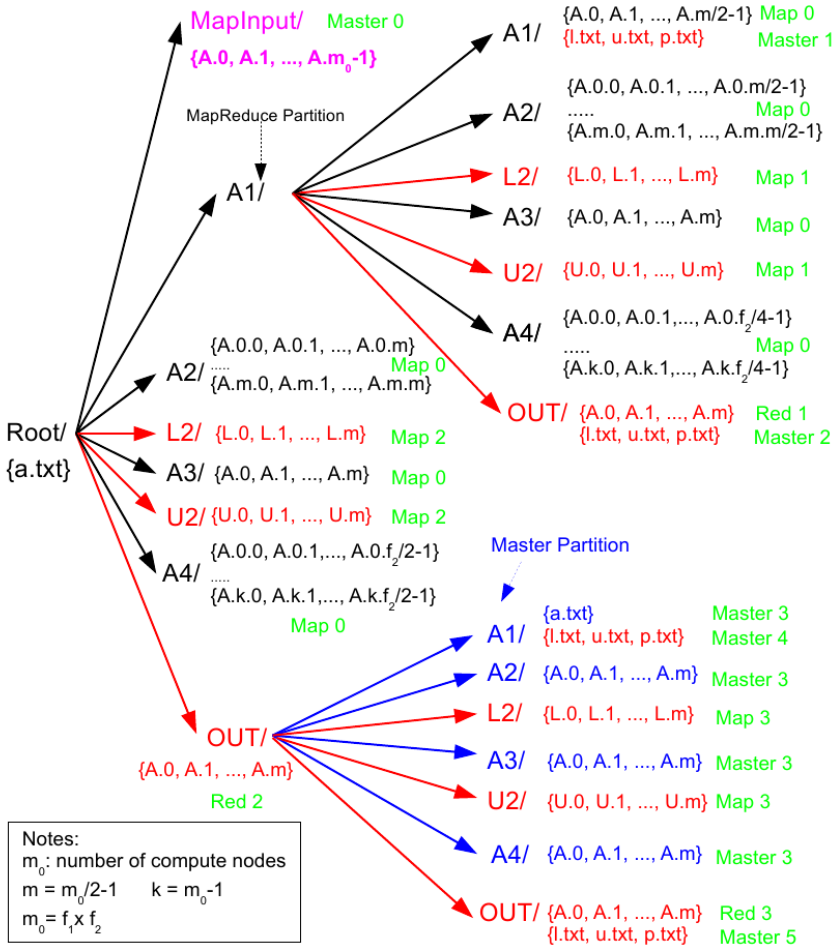
Número de Tareas

- m_0 es la cantidad de máquinas en el cluster
- n_b es el máximo tamaño de una matriz para computar en un solo nodo
- Número de tareas será $2^{\lceil \log_2 \frac{n}{n_b} \rceil}$:



(adaptado de Xiang Jingen, 2013)

Ejecución


MapInput/ Master 0
{A.0, A.1, ..., A.m0-1}
MapReduce Partition
A1/
A2/
L2/
A3/
U2/
A4/
OUT/
{A.0, A.1, ..., A.m/2-1} Map 0
{l.txt, u.txt, p.txt} Master 1
{A.0.0, A.0.1, ..., A.0.m/2-1}
.....
{A.m.0, A.m.1, ..., A.m.m/2-1}
Map 0
{L.0, L.1, ..., L.m} Map 1
{A.0, A.1, ..., A.m} Map 0
{U.0, U.1, ..., U.m} Map 1
{A.0.0, A.0.1,..., A.0.f2/4-1}
.....
{A.k.0, A.k.1,..., A.k.f2/4-1}
Map 0
{A.0, A.1, ..., A.m} Red 1
{l.txt, u.txt, p.txt} Master 2
Root/
{a.txt}
{A.0.0, A.0.1, ..., A.0.m}
.....
{A.m.0, A.m.1, ..., A.m.m}
Map 0
{L.0, L.1, ..., L.m} Map 2
{A.0, A.1, ..., A.m} Map 0
{U.0, U.1, ..., U.m} Map 2
{A.0.0, A.0.1,..., A.0.f2/2-1}
.....
{A.k.0, A.k.1,..., A.k.f2/2-1}
Map 0
Master Partition
{a.txt} Master 3
{l.txt, u.txt, p.txt} Master 4
{A.0, A.1, ..., A.m} Master 3
{L.0, L.1, ..., L.m} Map 3
{A.0, A.1, ..., A.m} Master 3
{U.0, U.1, ..., U.m} Map 3
{A.0, A.1, ..., A.m} Master 3
{A.0, A.1, ..., A.m} Red 3
{l.txt, u.txt, p.txt} Master 5
{A.0, A.1, ..., A.m}
Red 2
Notes:
m0: number of compute nodes
m = m0/2-1 k = m0-1
m0 = f1 x f2

Zinkevich et al., NIPS 2010

- Distribuir los datos al azar entre nodos
- Realizar descenso por el gradiente en cada nodo, por separado
- Unificar los resultados en un nodo central

Algoritmo en nodo worker

Algorithm 1 SGD$(\{c^1, \ldots, c^m\}, T, \eta, w_0)$

for $t = 1$ **to** T **do**
  Draw $j \in \{1 \ldots m\}$ uniformly at random.
  $w_t \leftarrow w_{t-1} - \eta \partial_w c^j(w_{t-1})$.
end for
return w_T.

(adaptado de Zinkevich et al, 2010)

Algoritmo en nodo central

Algorithm 2 ParallelSGD$(\{c^1, \dots c^m\}, T, \eta, w_0, k)$

for all $i \in \{1, \dots k\}$ **parallel do**
  $v_i = \text{SGD}(\{c^1, \dots c^m\}, T, \eta, w_0)$ on client
end for
Aggregate from all computers $v = \frac{1}{k} \sum_{i=1}^{k} v_i$ and **return** v

(adaptado de Zinkevich et al, 2010)

Dirección Alternada de Multiplicadores

- Optimizar dos funciones a la vez
  - Problema similar a regresión logística y SVMs
  - Alternar la optimización de una función y la otra
- Clave: en ciertas condiciones, la primera optimización se puede hacer en paralelo mientras que la segunda es reducida y puede hacerse en una sola máquina

$$\begin{aligned} \text{minimizar} \quad & f(x)+g(z) \\ \text{dado que} \quad & Ax+Bz=c \end{aligned}$$

- Ver Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers (201) por Boyd, Parikh, Chu, Peleato, Eckstein

- Más allá de MapReduce:
  - MPi: Message Passing Interface
  - AMQP: Advanced Message Queuing Protocol

MPI

- Estándar de facto
  - Desde 1994
  - Tres revisiones
  - Especificaciones independientes del lenguaje para llamadas de función
    - CORBA o RPC
    - Mucha menor latencia entre llamadas
  - Mismo programa en todos los nodos, se pasan datos entre ellos

MPI

- Provee:
  - Topología virtual
  - Sincronización
  - Comunicación
- Funciones:
  - Comunicación punto-a-punto (rendez-vous), send/receive
  - Topología Cartesiana o de grafo general
  - Combinar resultados parciales (gather/reduce)
  - Sincronización de nodos (barreras)
  - Información general (número de nodos, número de nodo actual, vecinos, topología)

Decisiones de diseño

- Todo el paralelismo es explícito
- Distintas implementaciones (en distinto hardware) sólo requieren recompilación del código

Implementación de ejemplo: MPICH

- Programas mpicc y mpiexec
- ssh con certificados (sin password)
- mpiexec -f lista-de-hostnames -n <número> programa-ejecutable
  - La lista de hostnames puede tener de manera opcional el número de cores después de un :
- mpiexec -f lista-de-hostnames -n 1 ./master : -n 15 ./slave
- Diferentes métodos de comunicación entre procesos
  - Más recient "némesis" usa memoria compartida para procesos en la misma máquina y sockets (o Myrinet-MX) para comunicación entre red

Ejemplo

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char ** argv) {
int rank, size;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
printf("I am %d of %d\n", rank + 1, size);
MPI_Finalize();
return 0;
}
```

Adapted from http://www.mcs.anl.gov/~balaji/permalinks/2014-06-06-argonne-mpi-basic.pptx

Diferencia con MapReduce

- MapReduce es un subconjunto del estándar MPI
  - Operaciones collectivas y operaciones de usuario
- MPI no es tolerante a fallas
- MPI provee muchas más primitivas
  - Requiere que el programador se ocupe de muchos más detalles
- En el caso de inversión de matrices, la implementación MPI es más eficiente para matrices pequeñas pero tiene mucho mayor overhead de comunicación
  - Escala de manera más pobre

Message Queues

- Message-Oriented Middleware (MOM)
  - Soporte por software o hardware
  - Envío y recepción de mensajes
  - Redes distribuidas heterogéneas
  - Normalmente asíncrono
    - Colas
    - Potencialmente persistentes (tolerancia a fallos)
  - Ruteo
  - Potencial transformación
- Requiere un message transfer
  - Talón de Aquiles

Advanced Message Queuing Protocol (AMQP)

- Estándar OASIS
- Características:
  - Orientación de mensajes
  - Colas
  - Ruteo
    - Punto-a-punto
    - Publish-and-subscribe
  - Confiabilidad
  - Seguridad
- Protocolo a nivel del cable: información en formato de octetos (protocolo binario)

Especificación AMQP

- Sistema de tipos
  - Tipos primitivos
  - Diccionarios
- Protocolo simétrico, asíncrono para enviar mensajes
- Un formato de mensajes estándar y extensible
- Un conjunto estándar pero extensible de "capacidades de mensajería"
- Semánticas de recepción:
  - at-most-once
  - at-least-once
  - exactly-once

ActiveMQ

- Proyecto de la fundación Apache
- Java
- Ejecutar el broker:
  - bin/activemq
  - Ejecutar programas clientes pasandole información sobre el host/puerto del broker
    - Normalmente vía variables de entorno

Ejemplo

```
String user = env("ACTIVEMQ_USER", "admin");
String password = env("ACTIVEMQ_PASSWORD", "password");
String host = env("ACTIVEMQ_HOST", "localhost");
int port = Integer.parseInt(env("ACTIVEMQ_PORT", "61616"));
String destination = arg(args, 0, "event");
ActiveMQConnectionFactory factory = new
ActiveMQConnectionFactory("tcp://" + host + ":" + port);
Connection connection = factory.createConnection(user, password);
connection.start(); Session session =
connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
Destination dest = new ActiveMQTopic(destination);
MessageProducer producer = session.createProducer(dest);
producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
TextMessage msg = session.createTextMessage(body);
```

AMQP vs. MapReduce

- El broker es el eslabón más débil
- Topología explícita en AMQP
  - Más versátil
  - Más trabajo por parte del programador
- Tolerancia a fallas de AMQP es potencialmente mejor que MapReduce
  - Cómputos intermedios pueden ser mantenidos en colas persistentes

UIMA Concepts

- Common Annotation Structure or CAS
  - Subject of Analysis (SofA or View)
  - JCas
- Feature Structures
  - Annotations
- Indices and Iterators
- Analysis Engines (AEs)
  - AEs descriptors

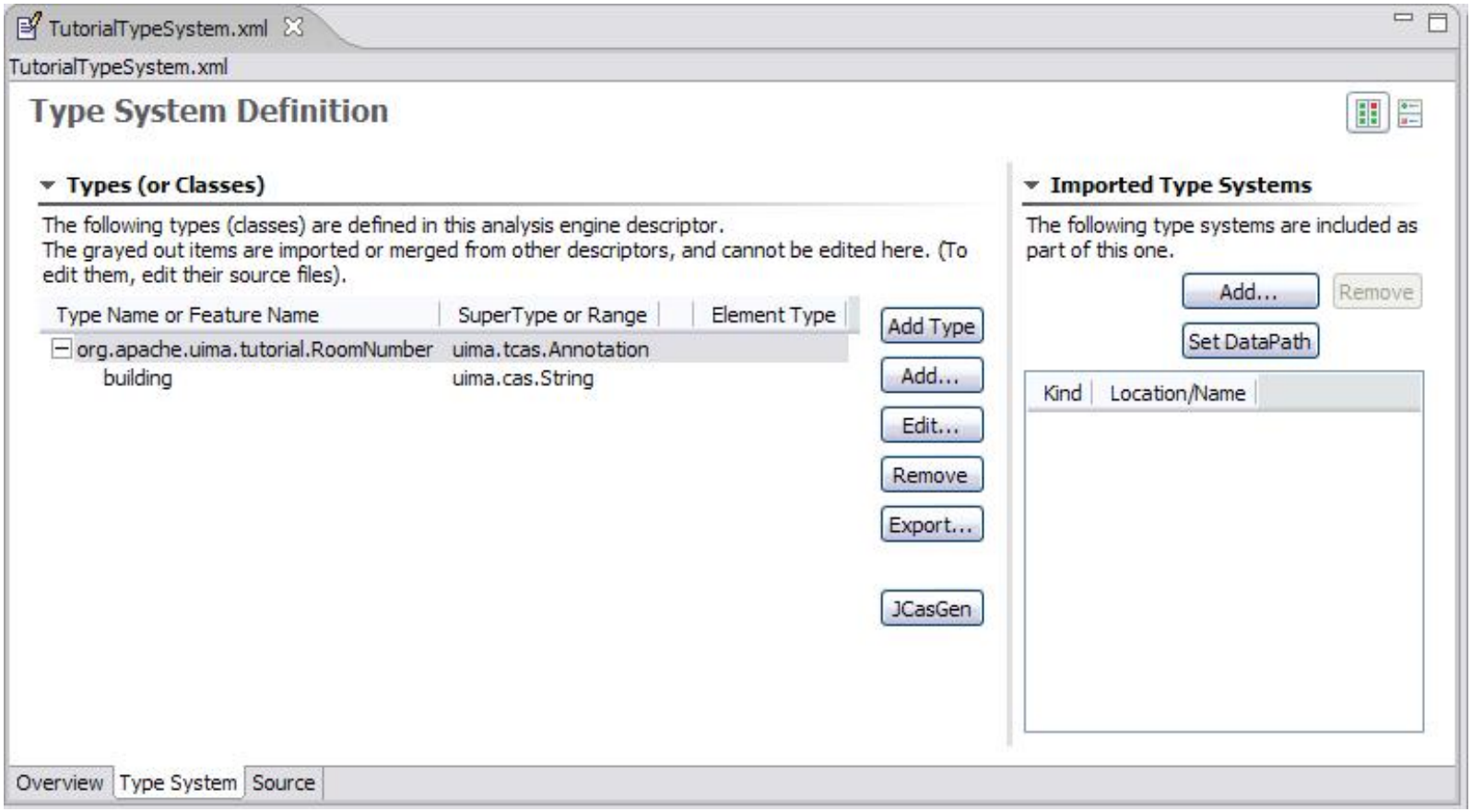
Room annotator

- From the UIMA tutorial, write an Analysis Engine that identifies room numbers in text.

  Yorktown patterns: 20-001, 31-206, 04-123 (Regular Expression Pattern: [0-9][0-9]-[0-2][0-9][0-9])
  Hawthorne patterns: GN-K35, 1S-L07, 4N-B21 (Regular Expression Pattern: [G1-4][NS]-[A-Z][0-9])

- Steps:
  1. Define the CAS types that the annotator will use.
  2. Generate the Java classes for these types.
  3. Write the actual annotator Java code.
  4. Create the Analysis Engine descriptor.
  5. Test the annotator.

Editing a Type System



The XML descriptor

```
<?xml version="1.0" encoding="UTF-8" ?>
  <typeSystemDescription xmlns="http://uima.apache.org/resourceSpecifier">
    <name>TutorialTypeSystem</name>
    <description>Type System Definition for the tutorial examples –
        as of Exercise 1</description>
    <vendor>Apache Software Foundation</vendor>
    <version>1.0</version>
    <types>
      <typeDescription>
        <name>org.apache.uima.tutorial.RoomNumber</name>
        <description></description>
        <supertypeName>uima.tcas.Annotation</supertypeName>
        <features>
          <featureDescription>
            <name>building</name>
            <description>Building containing this room</description>
            <rangeTypeName>uima.cas.String</rangeTypeName>
          </featureDescription>
        </features>
      </typeDescription>
    </types>
  </typeSystemDescription>
```

The AE code

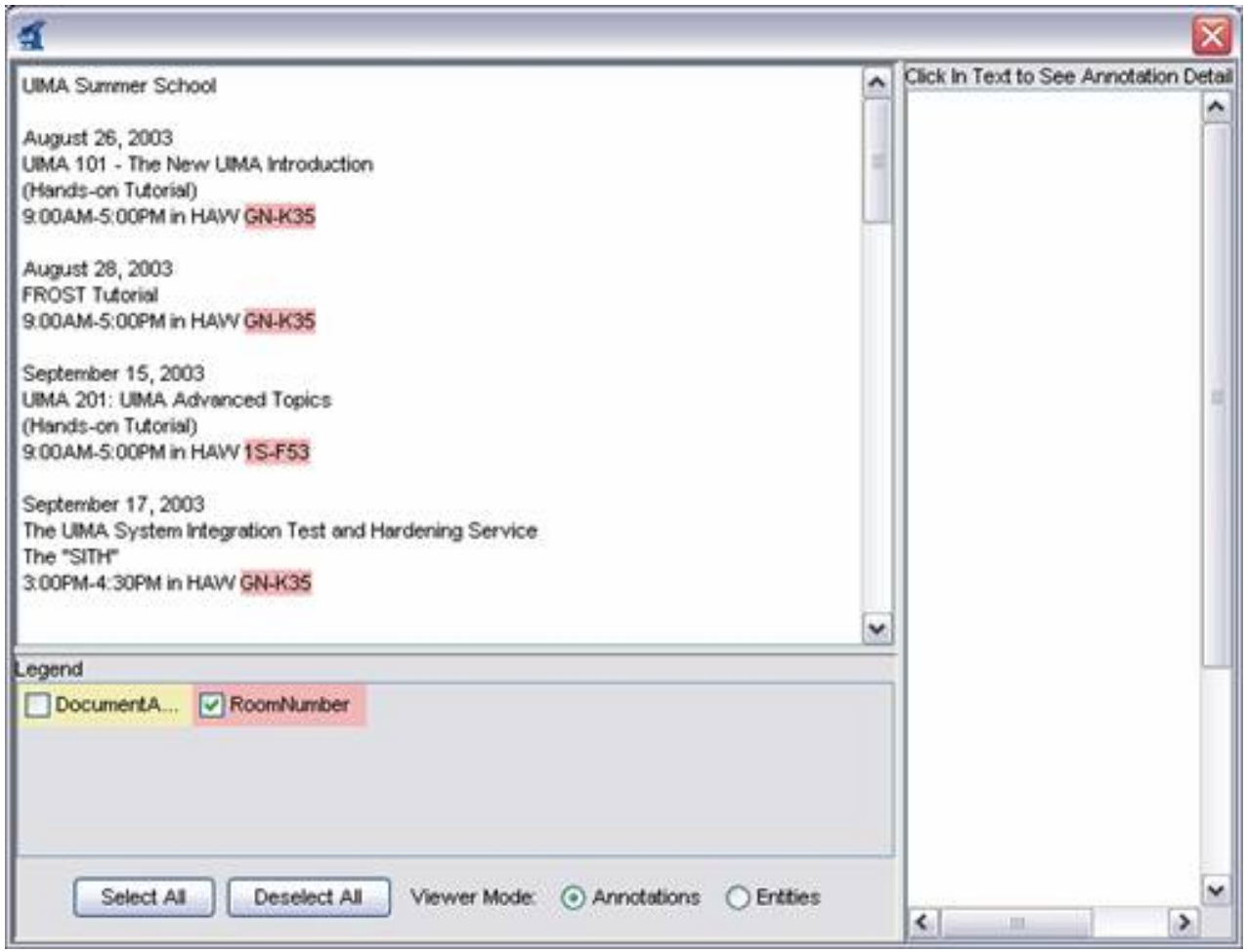
```
package org.apache.uima.tutorial.ex1;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

import org.apache.uima.analysis_component.JCasAnnotator_ImplBase;
import org.apache.uima.jcas.JCas;
import org.apache.uima.tutorial.RoomNumber;

/**
 * Example annotator that detects room numbers using
 * Java 1.4 regular expressions.
 */
public class RoomNumberAnnotator extends JCasAnnotator_ImplBase {
  private Pattern mYorktownPattern =
        Pattern.compile("\\b[0-4]\\d-[0-2]\\d\\d\\b");

  private Pattern mHawthornePattern =
        Pattern.compile("\\b[G1-4][NS]-[A-Z]\\d\\d\\b");

  public void process(JCas aJCas) {
    // next slide
  }
}
```

The AE code (cont.)

```
public void process(JCas aJCas) {
  // get document text
  String docText = aJCas.getDocumentText();
  // search for Yorktown room numbers
  Matcher matcher = mYorktownPattern.matcher(docText);
  int pos = 0;
  while (matcher.find(pos)) {
    // found one - create annotation
    RoomNumber annotation = new RoomNumber(aJCas);
    annotation.setBegin(matcher.start());
    annotation.setEnd(matcher.end());
    annotation.setBuilding("Yorktown");
    annotation.addToIndexes();
    pos = matcher.end();
  }
  // search for Hawthorne room numbers
  // ..
}
```

UIMA Document Analyzer

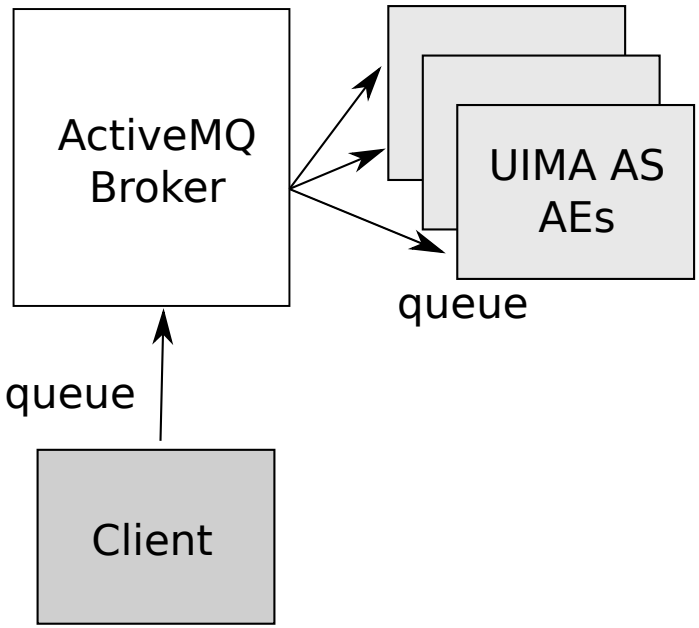
UIMA Document Analyzer (cont)



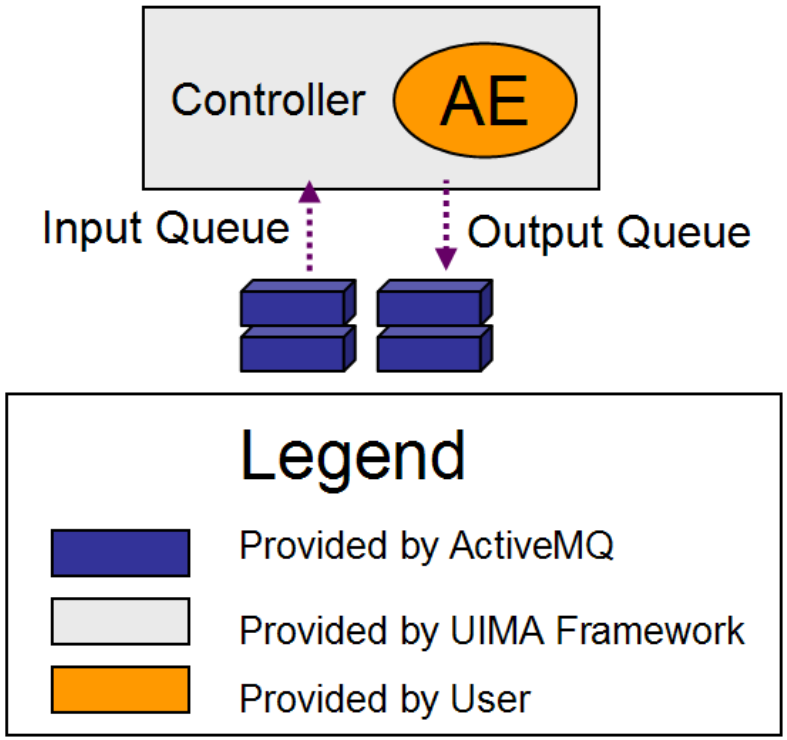
Custom Flow Controllers

- UIMA permite especificar cual AE deberá procesar la CAS en le paso siguiente, basado en las anotaciones que ya están en la CAS.
- `examples/descriptors/flow_controller/WhiteboardFlowController.xml`
  - FlowController que implementa un modelo de flujo simple de tipo "whiteboard" (pizarrón). Cada vez que recibe una CAS. se fija en el *pool* de AEs que todavía no han ejecutado sobre esa CAS y elije uno cuyos requerimientos de entrada ya hayan sido satisfechos.

UIMA AS: ActiveMQ



UIMA AS: Wrapping Primitive AEs



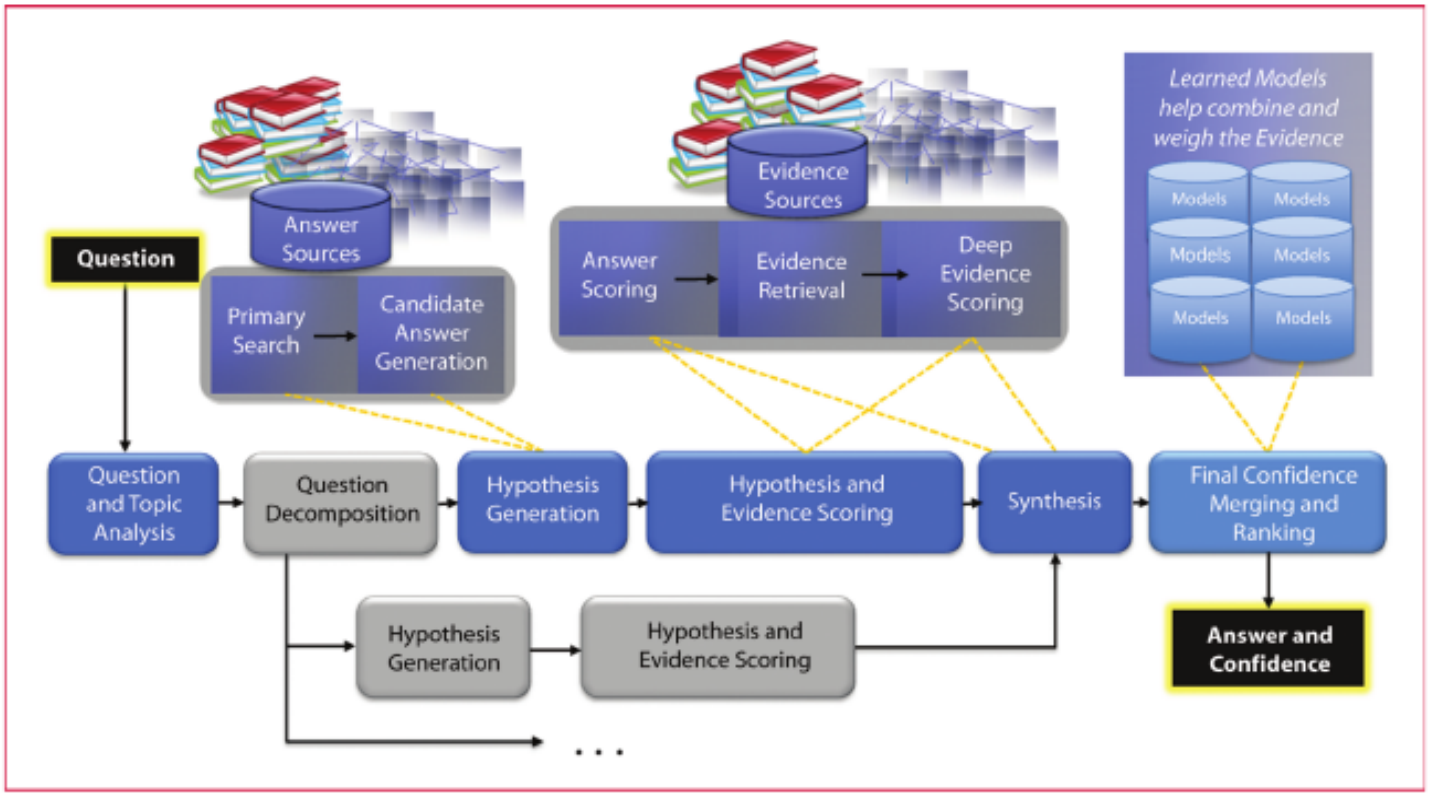
UIMA AS: Ventajas

- Muy flexible en térmios de dividir la carga de trabajo entres los nodos
  - Tienes control total sobre como dividir las colas en sub-colas, etc.
- Muy eficiente en términos de *overhead* en la red
  - Una CAS que va a ser dividida y procesada varias veces (en partes distintas) es enviada sólo una vez.
  - Sólo las anotaciones **requeridas** son enviadas y las anotaciones **nuevas** son devueltas.
    - Archivos de metadata (descriptores) son clave para que ésto funcione

UIMA AS: más información

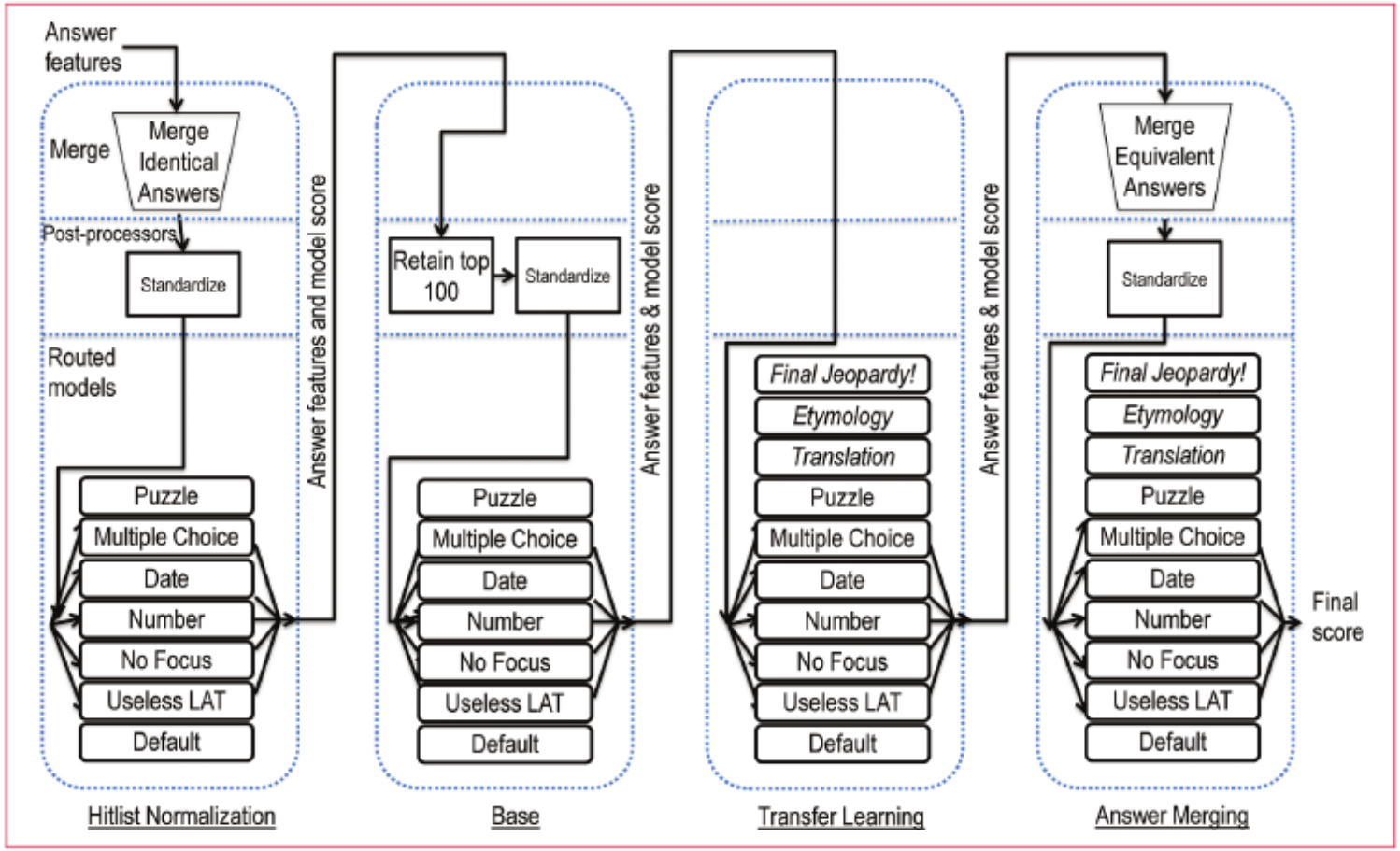
- http://uima.apache.org/doc-uimaas-what.html
- http://svn.apache.org/viewvc/uima/uima-as/trunk/README?view=markup
- http://uima.apache.org/d/uima-as-2.4.2/uima_async_scaleout.html

Arquitectura



DeepQA Architecture, adaptada de Ferrucci (2012)

First Four Phases of Merging and Ranking



de Gondek, Lally, Kalyanpur, Murdock, Duboue, Zhang, Pan, Qiu, Welty (2012)