12 JACA-DDM

JaCa-DDM es un sistema para diseñar, implementar, simular y ejecutar estrategias basadas en agentes y artefactos, para escenarios de **minería de datos distribuida** (DDM), es decir, casos donde los datos se encuentran en diferentes sitios. La organización del capítulo es como sigue: La sección 12.1 introduce el modelo abstracto de JaCa-DDM, definido a partir de los conceptos de estrategia y su sistema de despliegue. La sección 12.2 introduce la arquitectura de cómputo inducida por el modelo JaCa-DDM, describiendo en detalle los tipos de artefactos, programas de agente, detalles de configuración y el flujo de trabajo que el sistema provee. La sección 12.3 presenta un conjunto de estrategias incluidas en la distribución de JaCa-DDM. La sección 12.4 ejemplifica la instalación, configuración y uso del sistema. La sección 12.5 comenta algunos resultados experimentales que han sido obtenidos usando esta herramienta.

DDM

12.1 MODELO

El modelo de JaCa-DDM se define a partir de los conceptos de estrategia y su sistema de despliegue. Una **estrategia** define un flujo de trabajo a partir de un conjunto de agentes y artefactos y las interacciones que se dan entre estos, en el contexto de un proceso de minería de datos. El **sistema de despliegue** de una estrategia tiene que ver con aspectos de la distribución de los agentes y artefactos en los diferentes sitios de computo que se vayan a utilizar, así como su configuración. Formalmente:

Estrategia

Sistema de despliegue

Definición 12.1. *Una tupla $\langle Ags, Arts, Params, ag_1 \rangle$ denota una estrategia JaCa-DDM, donde:*

- $Ags = \{ag_1, \ldots, ag_n\}$ *es el conjunto de programas de agente definidos por el usuario como parte de la estrategia.*
- $Arts = \{art_1, \ldots, art_m\}$ *es el conjunto de tipos de artefacto definidos por el usuario como parte de la estrategia.*
- $Params = \{param_1 : tipo_1, \ldots param_k : tipo_k\}$ *es un conjunto de parámetros de la estrategia y sus tipos de datos asociados, donde* $tipo_{1,\ldots,k} \in \{int, bool, double, string\}$*. Los parámetros pueden afectar tanto a los agentes de la estrategia, como a sus artefactos.*
- $ag_1 \in Ags$ *es un programa de agente especial responsable de la estrategia. Para ello, debe satisfacer dos restricciones de diseño:*
  - *Contar con un plan para contender con un evento disparador* $+!start$*, que le indica que la estrategia puede ser lanzada.*

– *Detener la ejecución de la estrategia y enviar como resultado un mensaje* $\langle ag_0, tell, finish(ArtId)\rangle$*, donde* ag_0 *es el agente responsable del sistema de despliegue (Ver a continuación la definición 12.2) y* $ArtId$ *es el identificador del artefacto donde están almacenados los resultados del proceso.*

Aparte de estas restricciones, este agente puede hacer cualquier otra tarea a decisión del programador.

El flujo de trabajo exacto, la manera en que los agentes interactuan y usan sus artefactos, está encapsulado en los programas de agente que componen la estretegia. En la sección 12.3 usaremos diagramas de secuencia UML conjuntamente con esta definición, para describir un conjunto de estrategias incluídas en la distribución de JaCa-DDM.

Definición 12.2. *Una tupla* $\langle Nodos, DS, Arts, Estrat, Config, ag_0\rangle$ *denota un sistema de despliegue JaCa-DDM, donde:*

- $Nodos = \{nodo_0, nodo_1, \ldots, nodo_j\}$ *es el conjunto de nodos CArtAgO en los que se desplegará el sistema. Cada nodo puede ejecutar al menos un espacio de trabajo, donde se crean artefactos. Unicamente el* $nodo_0$ *ejecuta agentes Jason [1]. Cada nodo se denota por un par* $\langle Nodo, DirIP : Puerto\rangle$*.*
- $DS = \{ds_1, \ldots, ds_j\}$ *es un conjunto de fuentes de datos para cada nodo, exceptuando el* $nodo_0$*. Las fuentes de datos pueden definirse dinámicamente en tiempo de ejecución, o bien estáticamente en cada nodo.*
- $Arts = \{art_1, \ldots, art_i\}$ *es un conjunto de tipos de artefactos primitivos usados en el sistema de despliegue.*
- $Estrat$ *es una estrategia JaCa-DDM definida en términos de la definición 12.1.*
- $Config = \langle \delta, \pi\rangle$ *denota la configuración del sistema de despliegue, donde:*
  - $\delta = \langle (ag, nodo, i), \ldots\rangle$ *es un conjunto de asignaciones de* i *copias de un programa de agente* ag *focalizando en cierto nodo. Cuando el nodo es evidente, usamos la notación* $ag\#i$*.*
  - $\pi = \{(param : val), \ldots\}$ *es un conjunto de pares parámetro-valor, tal y como lo demanda la definición de la estrategia adoptada.*

Los detalles del despliegue están encapsulados en el agente ag_0. La manera en que estas definiciones conforman una arquitectura JaCa-DDM se explica en la siguiente sección.

12.2 ARQUITECTURA

Una estrategia JaCa-DDM se despliega en una arquitectura de cómputo distribuida como la que se muestra en la figura 12.1. Todos los agentes *ag* como

1 Esto pudiese parecer una limitante, pero más adelante será evidente que no lo es. Los artefactos, que son quienes tienen la carga computacional más pesada, si están distribuidos en los nodos del sistema. Los agentes pueden percibir/actuar sobre ellos de manera remota. Además, al estar todos los agentes en un mismo nodo, los costos de comunicación entre ellos es más bajo que en el caso en que también estuvieran distribuidos.

se mencionó, se ejecutan en el $nodo_0$; sin embargo, todos los nodos definen al menos un espacio de trabajo CArtAgO al cual los agentes pueden unirse lógicamente (*join*) para crear, percibir y usar artefactos *art*. Por lógicamente, entendemos que esto se hace explotando la invocación remota de métodos (RMI) provista por Java, de manera que aunque los agentes se ejecutan siempre en el $nodo_0$, pueden ejecutar operaciones de artefactos que se encuentran físicamente en otros nodos. Obseven que los nodos que no ejecutan a los agentes, tienen asociada una fuente de datos *ds*. De esta forma, los artefactos que ejecutan algoritmos de aprendizaje pueden ser distribuidos eficientemente, mientras que la comunicación entre agentes (local) se mantiene lo más rápida posible.

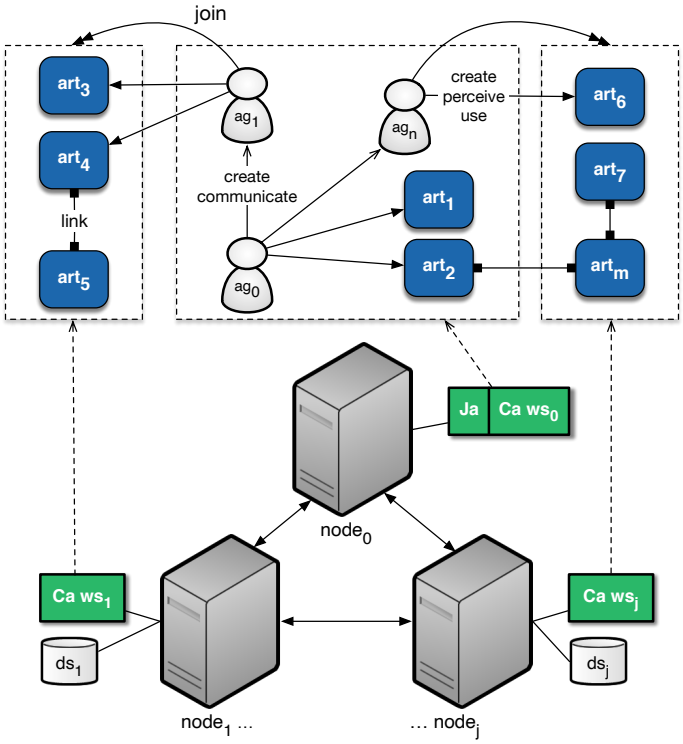

Figura 12.1: La arquitectura de JaCa-DDM.

Aunque JaCa-DDM fue concebida para la minería de datos distribuida, es posible explotar una computadora con multiples procesadores para ejecutar concurrentemente todos los espacios de trabajo en el $nodo_0$.

En lo que sigue, los programas de agente y tipos de artefactos primitivos provistos por JaCa-DDM son introducidos.

12.2.1 Artefactos

Los artefactos JaCa-DDM encapsulan clases de Weka [190], el conocido ambiente de minería de datos; y de MOA [11], su extensión para aprendizaje en línea masivo. Cualquier algoritmo de aprendizaje implementado en Java puede ser fácilmente encapsulado en un artefacto, la decisión de usar Weka/MOA es pertinente para efectos de comparación: Los resultados obtenidos por las estrategias JaCA-DDM pueden compararse con los resultados

obtenidos por los algoritmos centralizados, no basados en agentes, que Weka/MOA proveen.

El conjunto de tipos de artefactos provistos por JaCa-DDM se muestran en el cuadro 12.1. El usuario puede agregar otros artefactos al sistema de ser necesario, por ejemplo, nuevos clasificadores. Es importante considerar los siguientes líneamientos al definir nuevos artefactos:

- Los artefactos creados deben tener nombres únicos, aún si se encuentran en diferentes nodos de la arquitectura. Por convención, al crear un artefacto, el agente que lo ha creado incluye su nombre como prefijo. Por ejemplo: `worker78-classifierJ48`.
- Si un artefacto hace referencia a otro artefacto situado en un nodo distinto, es necesario que se registre en el directorio del sistema.

Tipo de Artefacto	Descripción
`Classifier`*XXX*	Una herramienta de clasificación, capaz de aprender nuevos modelos y usarlos para clasificar ejemplos, basada en Weka/MOA. *XXX* puede substituirse por alguno de los siguientes algoritmos: J48, VDST, Bagging, BaggingEnsemble.
`Directory`	Una herramienta de localización de servicios para agentes, artefactos y espacios de trabajo (al estilo páginas amarillas y blancas).
`Evaluator`	Una herramienta para la evaluación de modelos basada en Weka.
`GUI`	Una intefaz gráfica para configurar y lanzar experimentos.
`FileManager`	Una herramienta para escribir archivos ARFF, basado en Weka.
`InstancesBase`	Un repositorio de ejemplos de aprendizaje basado en Weka.
`LogBook`	Una herramienta para generar reportes de resultados de los experimentos.
`Oracle`	Una herramienta para distribuir conjuntos de entramiento centralizados en los diferentes nodos de la arquitectura, basado en Weka.
`TrafficMonitor`	Un sniffer para medir la carga de la red de la arquitectura, basado en utilidades de terceros. Su uso es opcional.
`Utils`	Una navaja suiza para los agentes, con multiples utilidades.

Cuadro 12.1: Tipos de artefactos provistos por JaCa-DDM.

12.2.2 Agentes

JaCa-DDM requiere de la definición de dos agentes especiales: El responsable de ejecutar experimentos ag_0, definido en el sistema de despliegue; y el responsable de la estrategia ag_1, definido en la estrategia misma. A continuación se detallan las funciones de estos dos agentes:

Responsable de experimentos

El agente ag_0, definido como parte del sistema de despliegue, se ejecuta al lanzar el sistema JaCa-DDM. Como responsable de ejecutar los experimentos propuestos, sus tareas incluyen:

CONFIGURAR EL EXPERIMENTO. La configuración de un experimento se lleva a cabo interactivamente, de acuerdo a la estrategia adoptada, gracias a un artefacto de tipo `GUI`, creado por ag_0. Como se especifica en la definición 12.2, esto incluye la distribución de los agentes (δ) y la inicialización de parámetros (π).

DISTRIBUIR DATOS DINÁMICAMENTE. Cuando JaCa-DDM se usa para simular escenarios distriuidos, ag_0 puede crear un artefacto de tipo `Oracle` para distribuir un conjunto de entrenamiento centralizado, en los diferentes nodos de la arquitectura. Esto es muy útil para comparar las estrategias distribuidas contra el caso centralizado (cuando este es viable). Alternativamente, las fuentes de datos pueden ya estar presentes en los nodos del sistema al inicio de la ejecución de JaCa-DDM y en ese caso, ag_0 no distribuye nada.

MONITOREO DE TRÁFICO. Es posible evaluar el tráfico de datos que genera la ejecución de un experimento, para ello ag_0 puede crear opcionalmente un artefacto de tipo `TrafficMonitor`.

DESPLIEGUE DE AGENTES. La creación e inicialización de los agentes Ags de la estrategia adoptada es responsabilidad de ag_0. La definición del sistema de despliegue 12.2, le indica cuantos agentes de cada tipo debe crear en los nodos de la arquitectura. La inicialización de estos agentes consiste en comunicarles información relevante para la ejecución de los experimentos, incluyendo:

- `node(NodeName)`. Cada agente conoce el nombre lógico del nodo que le ha sido asignado.
- `ipNode0(IPAdress:Port)`. Cada agente conoce la dirección IP y el puerto del $nodo_0$. Esta información se usa para utilizar artefactos que están ejecutándose en ese nodo, por ejemplo el directorio.
- `data(FilePath`. Cada agente conoce la ruta a los datos que usará en el proceso de aprendizaje.
- `param(ParamId,Val`. Cada agente conoce los parámetros de la estrategia que está siendo utilizada y sus valores.
- Planes primitivos. Se trata de un conjunto de planes genéricos, independientes de la estrategia adoptada, que sirven para registrar

artefactos en el directorio, intercambiar modelos entre artefactos, conocer cuantas copias de un tipo de agente dado hay en el sistema, etc.

EVALUAR MODELOS. A través de artefactos de tipo `Evaluator` y `LogBook`, ag_0 puede evaluar los modelos aprendidos y generar reportes sobre su desempeño. Un mensaje $\langle ag_1, tell, finish(ArtId)\rangle$, donde ag_1 es el agente responsable de la estrategia adoptada y $ArtId$ es el artefacto que almacena el modelo obtenido, le avisa que puede proceder a la evaluación.

LIMPIEZA. Si la evaluación de un experimento implica repeticiones iteradas del mismo, como en el caso de la validación cruzada, el ag_0 reinicia agentes y artefactos de acuerdo a los requerimientos del método de evaluación.

Responsable de estrategia

Recuerden que $ag_1 \in Estrat_{Ags}$ es el responsable de la estrategia, una especie de *concierge* que sirve de mediador entre los agentes de la estrategia y el agente ag_0 responsable de los experimentos. El agente ag_1 puede unirse a cualquier espacio de trabajo, dependiendo de los requerimientos de la estrategia adoptada. Sus responsabilidades incluyen:

INICIAR EL PROCESO DE APRENDIZAJE. Un mensaje $\langle ag_0, achieve, start\rangle$, le avisa a ag_1 que debe lanzar el proceso de aprendizaje.

FINALIZAR EL PROCESO DE APRENDIZAJE. Una vez finalizado el proceso de aprendizaje, ag_1 debe enviar un mensaje $\langle ag_0, tell, finish(ArtId)\rangle$, avisando al responsable de los experimentos ag_0 que el proceso ha finalizado y que el modelo obtenido se encuentra en el artefacto $ArtId$.

Además de estas tareas mínimas obligatorias, ag_1 puede llevar a cabo cualquier otra tarea, por ejemplo de inicialización o coordinación entre los agentes de la estrategia.

12.2.3 Configuración

El sistema de despliegue requiere de una configuración acorde con la definición 12.2. Hemos decidido que está configuración se represente en formato XML, de forma que un artefacto de tipo `GUI` pueda generarla, y alternativamente pueda ser fácilmente escrita o editada, por el usuario con un editor de texto. La configuración puede incluir:

- La estrategia *Estrat* adoptada. Al definirse una estrategia, debe generarse un descriptor XML acorde con la definición 12.1. Ejemplos de estos descriptores pueden encontrarse en los protocolos incluídos en la distribución en el directorio `sampleProtocols`.
- La dirección IP y los puertos usados por los nodos en *Nodos*.
- La distribución de agentes (δ) y la inicialización de parámetros (π).

- La distribución de datos *DS*. Esto puede hacerse de dos maneras:
  - Distribución estática de datos. Las fuentes de datos ya están distribuidas en los nodos de la arquitectura. Las fuentes pueden ser:
    * Un solo archivo, denotado por su nombre. Por ej.: `iris.arff`.
    * Multiples archivos, denotados por la raíz del nombre. Por ejemplo: `iris1.arff, iris2.arff`, etc.
  - Distribución dinámica de datos. Un conjunto de entrenamiento se encuentra en el $nodo_0$ y es distribuido en los demás nodos de la arquitectura, usando un artefacto de tipo `Oracle`. La distribución debe ser acorde a alguno de los siguientes métodos de evaluación:
    * `hold-out`. El conjunto de entrenamiento se parte en un conjunto de prueba y varios conjuntos de entrenamiento, de acuerdo con un parámetro que define el tamaño del conjunto de prueba, en términos de un porcentaje de ejemplos a usarse con este propósito. El resto de los ejemplos disponibles se distribuye de manera estratificada entre los nodos definidos.
    * `cross-validation`. Un parámetro indicando el número de pliegues, determina el radio de ejemplos usados para la prueba y los entrenamientos. Por ejemplo, un valor de cinco, genera cinco particiones del conjunto de entrenamiento original, en cada una de las cinco iteraciones que se harán, una de las particiones se reserva como conjunto de prueba y el resto se usan como conjunto de entrenamiento. Las particiones de entrenamiento se distribuyen en los nodos definidos en el sistema.

    Estos métodos de distribución de datos pueden usarse de manera conjunta, para ejecutar diferentes experimentos con exactamente la misma distribución de datos. La idea es ejecutar primero un experimento bajo distribución dinámica; y el resto de ellos con distribución estática, usando la distribución generada en el primer experimento.
- La evaluación del modelo que se adoptará en el experimento. Esto determina cuantas veces y bajo que modalidad, debe repetirse el experimento.

12.2.4 Flujo de trabajo

El flujo de trabajo de un experimento usando JaCa-DDM, suele ser el siguiente:

1. Un artefacto de tipo `GUI` es creado por ag_0. Este artefacto produce un configuración del sistema de despliegue en formato XML. Si tal configuración ya existe, es posible cargarla directamente a través de la misma interfaz. Una vez configurado el sistema de despliegue, ag_0 crea artefactos de los siguientes tipos, en el $nodo_0$: `Directory, Evaluator, LogBook, Utils` y opcionalmente `TrafficMonitor, Oracle`. También

crea artefactos de tipo `FileManager` y opcionalmente `TrafficMonitor` en el resto de los nodos del sistema. Posteriormente ag_0 computa la distribución de datos, de acuerdo a la modalidad elegida. Al terminar, ag_0 crea los agentes $Estrat_{Ags}$ y los inicializa, asignando nodos conforme a la configuración del sistema de despliegue. Este paso puede visualizarse en la figura 12.2.

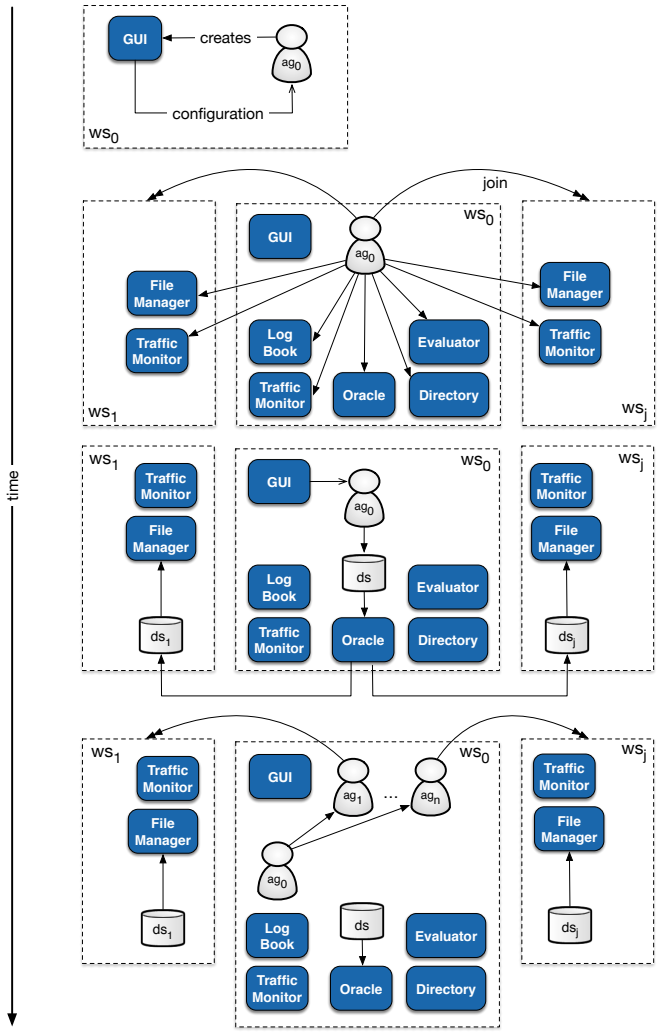

Figura 12.2: Paso 1 del flujo de trabajo de un experimento JaCa-DDM.

2. Un mensaje $\langle ag_i, tell, ready \rangle$ es enviado a ag_0 cada vez que un agente $ag_{i=1,\ldots,n}$ se une a su espacio de trabajo, anunciando que están listos para comenzar el proceso de aprendizaje.

3. Un mensaje $\langle ag_0, achieve, start \rangle$ es enviado al agente ag_1 para que lance el proceso de aprendizaje. El agente responsable de la estrategia hará todo lo necesario para lanzar el proceso este proceso y terminarlo con éxito.

4. Al terminar el proceso de aprendizaje, un modelo ha sido computado y almacenado en un arterfacto $ArtId$. Un mensaje $ag_1, tell, finish(ArtId)$ es enviado a ag_0 para informarle que el proceso de aprendizaje ha ter-

minado y donde puede encontrar el modelo computado. La figura 12.3 ilustra los pasos tres y cuatro del flujo de trabajo.

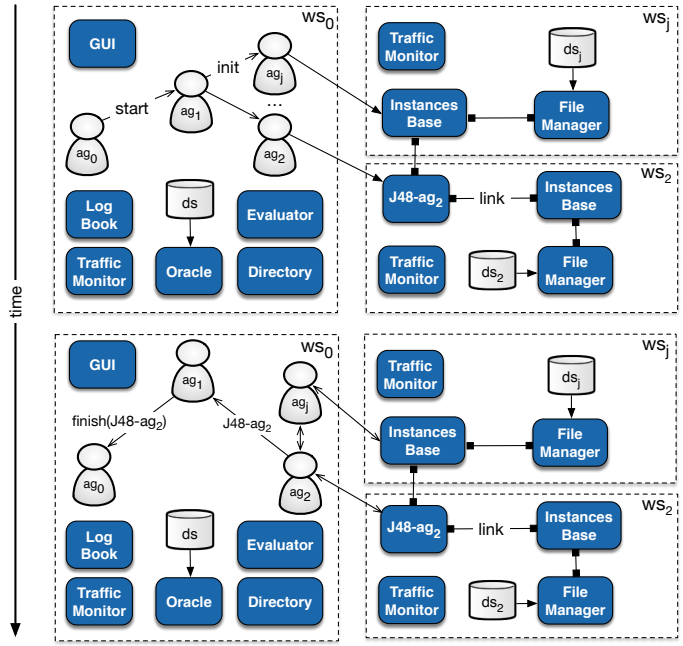

Figura 12.3: Pasos 3 y 4 del flujo de trabajo de un experimento JaCa-DDM.

5. Cuando ag_0 recibe el mensaje que le anuncia la finalización del proceso de aprendizaje, recupera el modelo computado para su evaluación. Los resultados obtenidos los despliega en la interfaz gráfica provista por el artefacto de tipo `GUI`. La figura 12.4 ilustra este paso.

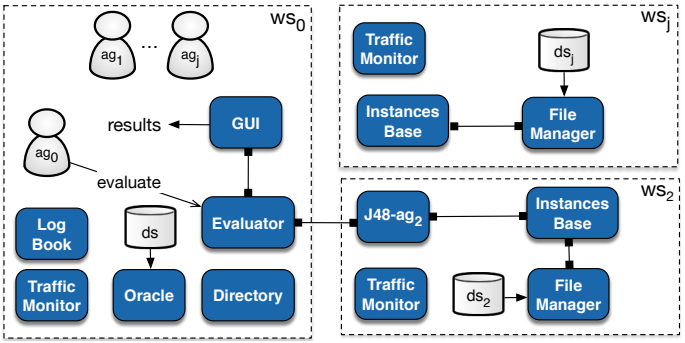

Figura 12.4: Paso 5 del flujo de trabajo de un experimento JaCa-DDM.

6. En cada nodo, ag_0 borra los agentes y artefactos creados y ejecuta una nueva iteración del experimento, si esto es necesario.

12.3 ESTRATEGIAS

12.4 INSTALACIÓN Y USO

Es posible descargar el sistema en la siguiente dirección:

https://sourceforge.net/projects/jacaddm/files/latest/download

La distribución de JaCa-DDM incluye tres directorios, a saber:

- `node0`: Contiene las fuentes, librerías y ejecutables relacionados con el $nodo_0$ del modelo JaCa-DDM. Puede verse como el directorio principal del sistema, donde la interfaz gráfica de configuración está definida. Para ejecutar la interfaz en sistemas operativos estilo UNIX ejecutar:

```
> ./run.sh
```

  Solo es necesario tener este directorio en la computadora que controlará el experimento de DDM.

- `defaultNode`: Contiene las fuentes, librerías y ejecutables relacionados con los nodos $nodo_1, \ldots, nodo_n$ del modelo JaCa-DDM. Estos nodos representan lógicamente un sitio donde tenemos datos que serán usados en el proceso de DDM. Es posible definir varios sitios en una sola computadora y ejecutar JaCa-DDM en una **modalidad concurrente**. Si el experimento es realmente distribuido, es necesario copiar este directorio en cada una de las computadoras donde vayamos a definir sitios. Es más natural pensar en escenarios distribuidos con un sitio por computadora disponible. En ambos casos el nodo se ejecuta de la siguiente manera:

Modalidad concurrente

```
> ./run.sh localhost 8080
```

  donde 8080 puede ser substituido por cualquier puerto IP disponible. Cada sitio necesita un puerto IP único.

- `sampleProtocols`: Contiene varios directorios, cada uno de ellos definiendo las diversas estrategias que se han definido con anterioridad. Los archivos XML definen las estrategias para su uso en JaCa-DDM. Solo se necesita una copia de este directorio en el $nodo_0$

Por ejemplo, en la siguiente sesión he levantado dos nodos mi computadora, usando los puertos 8080 y 8081.

```
defaultClient> ./run.sh localhost 8080 &
[1] 1046
CArtAgO Node Ready on localhost:8080
defaultClient> ./run.sh localhost 8081 &
[2] 1057
CArtAgO Node Ready on localhost:8081
```

Si el experimento será ejecutado de manera distribuida, es necesario verificar que la configuración de la red y su seguridad sean los correctos. Básicamente, que los puertos a utilizar estén abiertos y a disposición de nuestro sistema.

En la siguiente he levantado en $nodo_0$, accediendo a la interfaz gráfica de JaCa-DDM, que se muestra en la figura 12.5.

```
node0> ./run.sh &
[1] 1161
```

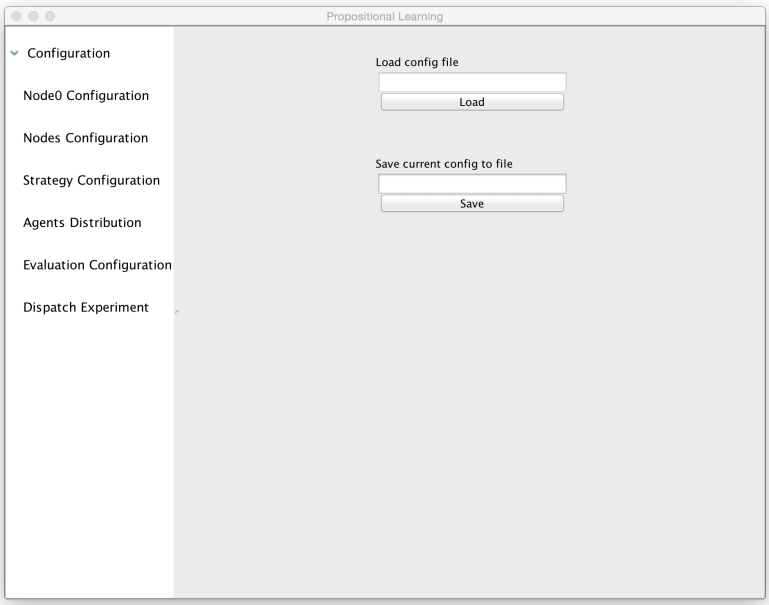

Figura 12.5: La ventana principal de JaCa-DDM

Los scripts `run.sh` de `node0` y `defaultNode` pueden leerse con un editor de texto. Lo que hacen es ejecutar el sistema multiagente y cliente por default respectivamente. Pare ello, invocan a `java` con el `classpath` adecuado en cada caso. Las librerías que se usarán, están incluidas en la carpeta `lib` de ambos componentes del sistema.

12.4.1 Configuración y ejecución de un experimento

El menú a la izquierda de la interfaz gráfica está constituído por una serie de pasos que, en orden descendente, nos permitirá configurar y ejecutar un experimento. La primer entrada nos permite, cargar una configuración previa y guardar en disco una configuración establecida con la interfaz. Las configuraciones se guardan en formato XML. Los aspectos a configurar se detallan a continuación.

Configuración del $nodo_0$

Es necesario establecer la dirección IP del $nodo_0$. Al hacer doble clic en menú `Node0 Configuration`, el sistema nos mostrará tal dirección (la de `localhost` en nuestro caso) y nos presentará un botón para guardar esta información.

Configuración de los demás nodos

Es necesario proveer la misma información para los demás nodos en el sistema y darles un nombre. Al hacer doble clic en el menú `Nodes Configuration`, podremos llevar a cabo está tarea. La interfaz se muestra ahora como en la figura 12.6.

Tres elementos deben definirse para cada nodo en el sistema:

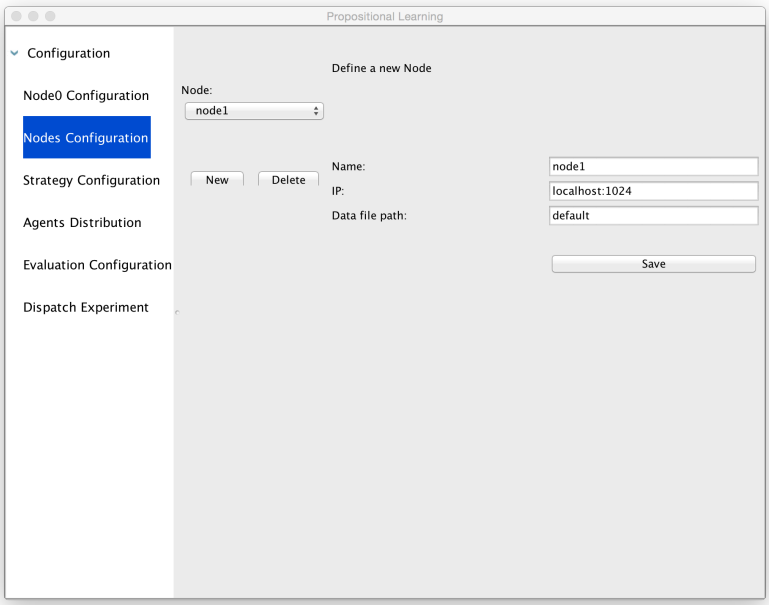

Figura 12.6: La configuración de los nodos $1, \ldots, n$ en JaCa-DDM

- Nombre. Un identificador para el nodo.
- IP. Un par de forma `direcciónIP:puerto`.
- Ruta a archivo de datos. La ruta al directorio donde se almacenarán los datos de entrenamiento. En los sistemas UNIX, puede usarse algo como `/tmp/dataN`, donde N es el número del nodo (Aunque estrictamente, cualquier nombre puede usarse). La única restricción importante aquí es que el sistema tenga permisos de escritura en ese directorio.

Una vez que se ha especificado la información para un nodo, hay que darle clic al botón `New` para agregarlo a la lista de nodos disponibles. Estos se listan el un menú de cascada en la parte superior de la interfaz gráfica. Es posible editar/eliminar un nodo, seleccionándolo en ese menú. He llamado `nodo1` y `nodo2` a los nodos que estamos ejecutados en los puertos 8080 y 8081, respectivamente. El botón `Save` guarda los cambios realizados en el archivo de configuración correspondiente.

Configuración de una estrategia

El menú `Strategy Configuration` nos permite adoptar una estrategia y configurarla. El directorio `sampleProtocols` incluye un conjunto de estrategias predefinidas en JaCa-DDM, mismas que han sido descritas en la sección 12.3. Utilizaré `centralizing` como ejemplo [2]. Para ello he cargado el archivo `centralizing.xml`, lo que determina los parámetros a inicializar. La figura 12.7 muestra la configuración para este caso.

2 Este mismo ejemplo puede encontrarse en el archivo README de la distribución de JaCa-DDM.

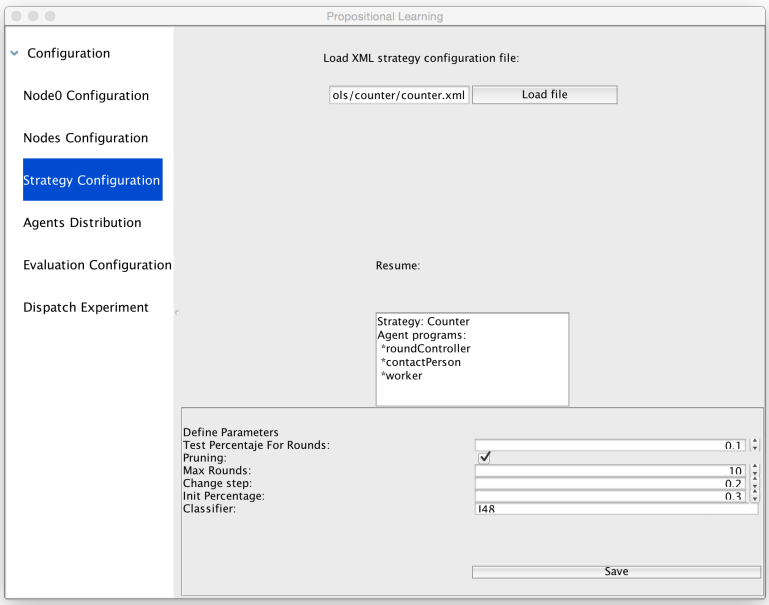

Figura 12.7: La configuración de una estrategia (counter) en JaCa-DDM

El clasificador `VFDT` es un algoritmo de inducción de árboles de decisión incremental. Se usará poda en la construcción de los árboles. Otros clasificadores requerirán diferentes parámetros.

12.4.2 Configuración de la distribución de agentes

El menú `Agents Distribution` nos permite especificar la distribución de agentes δ del sistema de despliegue. La opción `Refresh`, actualiza la interfaz para mostrarnos los tipos de programa de agente (dependiente de la estrategia) y nodos disponibles. La figura 12.9, muestra la distribución de agentes para este ejemplo.

Se han creado dos agentes `sender`, uno en cada nodo del sistema; y un agente `contactPerson` en el $nodo_1$.

12.4.3 Configuración del método de evaluación

Cuatro métodos de evaluación son propuestos, conforme a lo descrito en la sección 12.2.3. En este caso usaremos *cross-validation* con la configuración que se muestra en la figura **??**.

En este caso, haremos una validación cruzada de diez pliegues con una repetición. Los datos de los experimentos se guardaran en `/tmp/test`. Al igual que en el caso de los datos de entrenamiento, este debe ser un escritorio donde el sistema tenga permisos de escritura.

Es conveniente guardar la configuración en la opción `Configuration`, con lo que se generará un archivo XML con toda la información de configuración requerida por JaCa-DDM. Para este ejemplo el archivo luce así:

```
1 <?xml version="1.0"?>
```

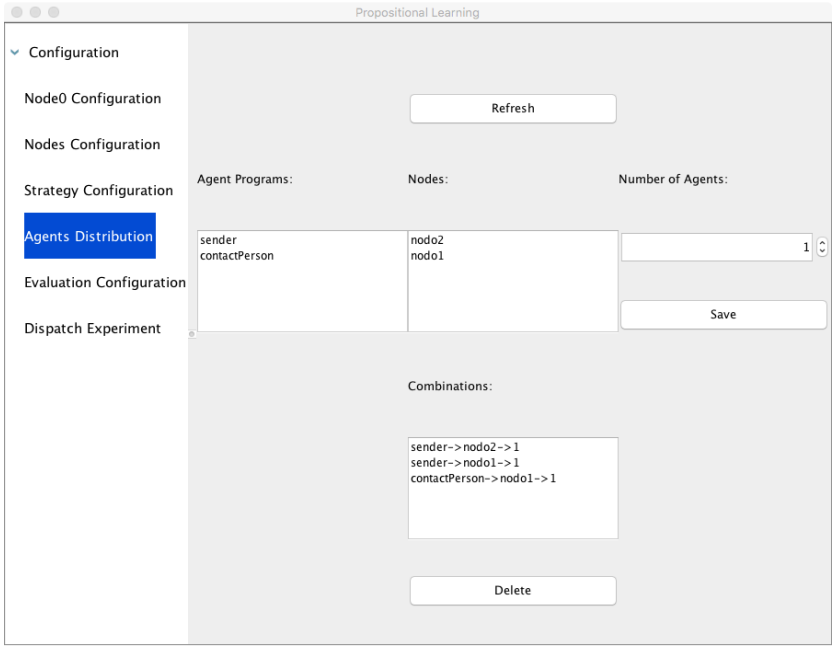

Figura 12.8: La distribución de agentes en JaCa-DDM

```
<config xmlns="http://uv.mx/hlimon"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://uv.mx/hlimon sys.xsd">

<name>Centralizing</name>

<agents>
  <agent>
    <program>contactPerson</program>
    <file>../sampleProtocols/centralizing/contactPerson.asl</file>
  </agent>
   <agent>
    <program>sender</program>
    <file>../sampleProtocols/centralizing/sender.asl</file>
  </agent>
</agents>

<params>
  <param>
    <name>Pruning</name>
    <type>boolean</type>
  </param>
    <param>
    <name>Classifier</name>
    <type>string</type>
  </param>
</params>

</config>
```

12.4.4 Ejecutando el experimento

La opción `Dispatch Experiment` permite lanzar el experimento configurado en los pasos anteriores. El botón `Refresh` muestra el listado de las opciones

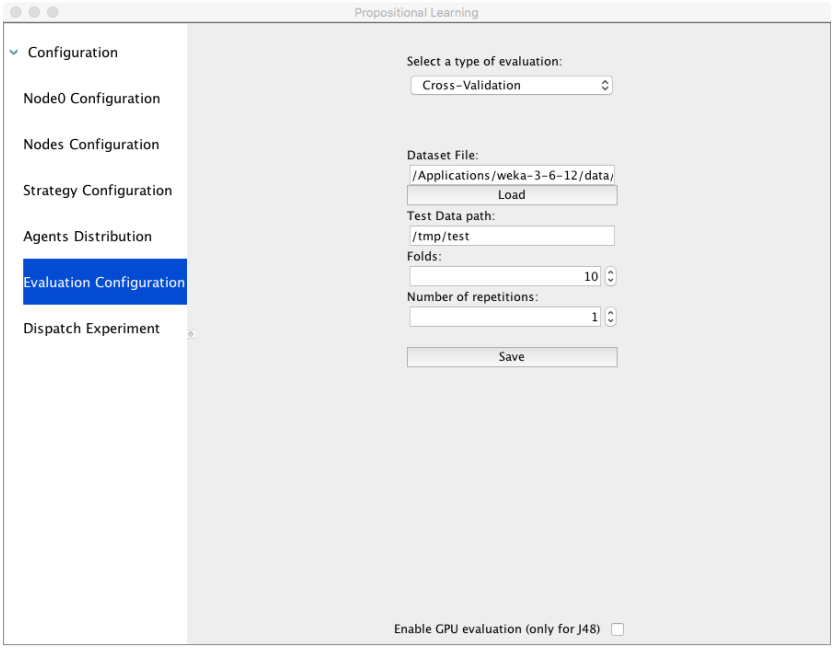

Figura 12.9: El método de evaluación *cross-validation* en JaCa-DDM

de configuración con las que se ejecutará el experimento. El botón `Start` ejecuta el experimento. El resultado, en este caso, puede verse en la figura 12.10.

En la ventana de resultados se muestra la salida global (promedios de los diez pliegues del experimento) y la matriz de confusión asociada. En la misma ventana se puede consultar los datos de cada iteración del experimento.

12.5 RESULTADOS EXPERIMENTALES

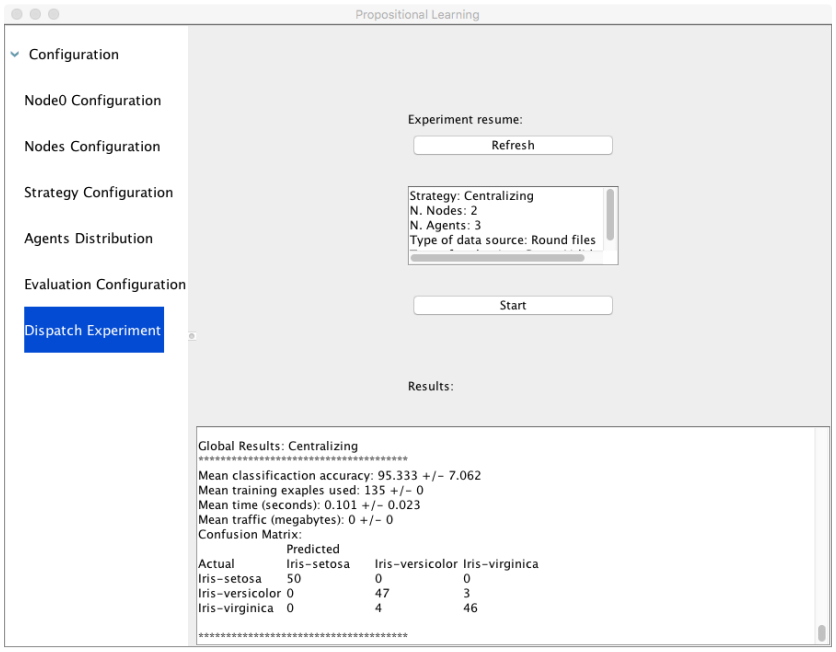

Figura 12.10: El método de evaluación *cross-validation* en JaCa-DDM