



**INSTITUTO POLITECNICO NACIONAL**  
**CENTRO DE INVESTIGACION EN COMPUTACION**



**“DESARROLLO DE UN SISTEMA GRID COMPUTING  
PARA LA DISTRIBUCION DE SERVICIOS EN  
LABORATORIOS VIRTUALES”**

**TESIS**

**QUE PARA OBTENER EL GRADO DE:  
MAESTRO EN CIENCIAS DE LA COMPUTACION**

**PRESENTA:  
ING. JOSUE RANGEL GONZALEZ**

**DIRECTOR DE TESIS: DR. LUIS ALFONSO VILLA VARGAS**  
**DIRECTOR DE TESIS: DR. JOSE LUIS OROPEZA RODRIGUEZ**

MÉXICO D.F. JUNIO DEL 2009



# INSTITUTO POLITECNICO NACIONAL

## SECRETARIA DE INVESTIGACIÓN Y POSGRADO

### ACTA DE REVISIÓN DE TESIS

En la Ciudad de  México, D.F.  siendo las  14:00  horas del día  4  del mes de  Junio  de  2009  se reunieron los miembros de la Comisión Revisora de Tesis designada por el Colegio de Profesores de Estudios de Posgrado e Investigación del:

#### **Centro de Investigación en Computación**

para examinar la tesis de grado titulada:

#### **“DESARROLLO DE UN SISTEMA GRID COMPUTING PARA LA DISTRIBUCIÓN DE SERVICIOS EN LABORATORIOS VIRTUALES”**

**RANGEL**

Apellido paterno

**GÓNZALEZ**

materno

**JOSUE**

nombre(s)

Con registro:

A	0	7	0	2	4	3
---	---	---	---	---	---	---

aspirante al grado de: **MAESTRÍA CIENCIAS DE LA COMPUTACIÓN**

Después de intercambiar opiniones los miembros de la Comisión manifestaron **SU APROBACIÓN DE LA TESIS**, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

#### LA COMISIÓN REVISORA

**Presidente**

Dr. Ricardo Barrón Fernández

**Secretario**

M. en C. Alejandro Botello Castillo

**Primer vocal  
(Director de tesis)**

Dr. Luis Alfonso Villa Vargas

**Segundo vocal  
(Director de tesis)**

Dr. José Luis Oropeza Rodríguez

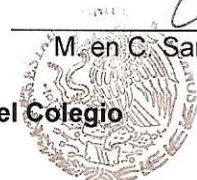
**Tercer Vocal**

M. en C. Felipe Dávalos Rodríguez

**Suplente**

M. en C. Sandra Dinora Orantes Jiménez

**El presidente del Colegio**



Dr. Jaime Álvarez Gallegos

DIRECCION



**INSTITUTO POLITECNICO NACIONAL**  
**SECRETARÍA DE INVESTIGACIÓN Y POSGRADO**

**CARTA CESION DE DERECHOS**

En la Ciudad de México D.F. el día 12 del mes Junio del año 2009, el que suscribe Josué Rangel González alumno del Programa de Maestría en Ciencias de la Computación con número de registro A070243, adscrito al Centro de Investigación en Computación, manifiesta que es autor intelectual del presente trabajo de Tesis bajo la dirección del Dr. Luis Alfonso Villa Vargas y del Dr. José Luis Oropeza Rodríguez y cede los derechos del trabajo intitulado Desarrollo de un Sistema Grid Computing para la Distribución de Servicios en Laboratorios Virtuales, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección [joshuargmx@hotmail.com](mailto:joshuargmx@hotmail.com). Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Josué Rangel González

Nombre y firma



---

## RESUMEN

Debido a la gran popularidad de los ambientes de trabajo en red, los emergentes procesadores multi-core paralelos y computación distribuida se requiere el desarrollo de aplicaciones en todos los niveles, además de ambientes de computación para las PC en escala-Internet como lo son las Grid y P2P.

Aplicaciones como los laboratorios virtuales pueden ofrecer una gran ayuda a todas aquellas personas que carecen de infraestructura adecuada y recursos económicos para poder comprar equipos costosos con los cuales se pueden ejecutar pruebas hacia un producto que van a lanzar al mercado. Desgraciadamente estos laboratorios no siempre cuentan con una cantidad infinita de recursos para que una gama de múltiples usuarios puedan hacer uso de sus servicios al mismo tiempo, de esta manera se presentan problemas, ya que la demanda de recursos puede llegar a ser excesiva para un solo servidor, aunque este cuente con multiprocesamiento y/o procesamiento paralelo, por tanto se da paso a una nueva forma de atacar tales problemas mediante la tecnología que ofrece el Grid Computing, está una múltiples computadoras heterogéneas para ofrecer una mayor cantidad de recursos, además de ello brinda algoritmos de distribución de carga y balanceo de trabajo para ofrecer una mejor capacidad de computo.

Es así como surge esta investigación, que propone un sistema Grid Computing basado en .NET para la distribución de servicios en laboratorios virtuales donde nos enfocamos específicamente al servicio de simulación antes de que un producto sea lanzado al mercado, este tipo de tareas requiere un soporte avanzado para múltiples tareas con relaciones entre las dependencias de datos, algunas de las simulaciones requieren de múltiples módulos, los cuales reciben entradas de datos, realizan operaciones y generan resultados, por se ofrece una solución para hacer frente a este tipo de problemas.

Algunas de las contribuciones de este trabajo son la reducción de tiempo de espera de los usuarios al realizar simulaciones de sistemas complejos, la utilización de equipos de escritorio heterogéneos para la generación de una Grid.



---

## ABSTRACT

Due to the popularity of the work environment on a network, the emerging multi-core processors, parallel and distributed computing requires the development of applications at all levels, besides computing environments for Internet-scale computer as are the Grid and P2P.

Applications such as virtual laboratories can provide a great help to all those who lack adequate infrastructure and financial resources to purchase expensive equipment which can be run to test a product that will launch to market. Unfortunately, these laboratories do not always have an infinite quantity of resources for a range of multiple users to use their services at the same time, this would present problems, since the demand for resources may be excessive for a single server, although this has multiprocessing and / or parallel processing, thus giving way to a new way to tackle such problems through technology offered by the Computing Grid is heterogeneous joins multiple computers to provide more resources, plus It provides algorithms for load distribution and balancing provide a better ability to compute.

Thus arises this research, which proposes a system based on Grid Computing. NET for the distribution of services in virtual laboratories where we focus specifically to simulation before a product is marketed, such tasks require a support advanced multi-tasking relationships among units of data, some models require multiple modules, which receive data inputs, and generate operating results, provides a solution to address such problems.

Some of the contributions of this work are the reduction of waiting time for users to perform simulations of complex systems, the use of heterogeneous desktops for the generation of a Grid.



---

## AGRADECIMIENTOS

### *A mis padres. Rogelio y Carmen*

Por su apoyo incondicional en todo momento, por sus consejos y regaños ya que con cada uno de ellos me hicieron madurar y me ayudaron a darme cuenta que la vida hay que vivirla paso a paso. Gracias por su confianza y Amor

### *A mi hermano. Adrian*

Por creer en mí y por ayudarme en los momentos difíciles de mi vida, gracias por demostrarme que siempre tendré tu apoyo.

### *A mi esposa. Nancy*

Por brindarme los momentos más hermosos de mi vida, por tu fuerza, alegría y cariño, el cual siempre te encargas de transmitirme, por crear en mi una sonrisa hasta en los momentos más difíciles, por tu apoyo y por dejarme compartir toda mi vida a tu lado, sobre todo por el Amor que me brindas día con día.

### *A mi hija. Melanie*

Ya que con tu llegada a nuestras vidas nos haz brindado la alegría más grande del mundo, porque desde este momento eres para nosotros el motivo más importante para seguir adelante persiguiendo nuevas metas.

### *A mi primo. Ismael*

Por tus consejos, ayuda académica y sobre todo por ser un buen amigo que nunca me diste la espalda cuando necesitaba de tu ayuda, por no odiarme ya que siempre te ganaba cuando jugábamos tenis de mesa.

### *A mis familiares y amigos*

Por tenerme paciencia en todo momento, ya que en muchas ocasiones deje de brindarles tiempo en momentos de convivencia, por creer en mí y por compartir todos aquellos momentos significativos en mi vida.





---

***A mis asesores. Dr. Luis Alfonso Villa Vargas y José Luis Oropeza Rodríguez***

Por todo el tiempo que me brindaron ayudándome o corriéndome en el desarrollo de este proyecto de investigación, por sus consejos y ayuda.

***Al personal de la UTE***

En especial a Elda y a Silvia, porque siempre me atendieron con una sonrisa y me trataron como a un amigo, muchas gracias por sus atenciones.

Al CONACYT y a la SIP por el apoyo brindado para la realización de mis estudios. Al Instituto Politécnico Nacional por brindarme la oportunidad de realizar un postgrado de calidad. Gracias Centro de Investigación en Computación por darme un mundo de conocimiento y por ser mi hogar durante más de 2 años.



## INDICE

<b>ÍNDICE FIGURAS .....</b>	<b>VIII</b>
<b>ÍNDICE TABLAS .....</b>	<b>IX</b>
<b>GLOSARIO .....</b>	<b>X</b>
<b>CAPITULO 1. INTRODUCCIÓN .....</b>	<b>1</b>
1.1 Plataformas virtuales .....	2
1.2 La educación virtual: ¿ una alternativa a la educación tradicional? .....	2
1.3 Laboratorios virtuales.....	4
1.4 Uso de laboratorios virtuales en áreas de la ciencia .....	5
1.5 Planteamiento del problema.....	5
1.6 Justificación .....	6
1.7 Objetivos.....	7
1.7.1 Objetivo general .....	7
1.7.2 Objetivos particulares .....	7
1.8 Alcances .....	8
1.9 Limitaciones .....	8
1.10 Estructura de la tesis.....	9
<b>CAPITULO 2. MARCO TEORICO .....</b>	<b>10</b>
2.1 Historia de la computación en malla .....	11
2.2 Presentación de trabajos relacionados con este proyecto de investigación .....	11
Planificación estática de programas de flujo de datos síncronos para procesamiento de señales digitales .....	11
Un área de trabajo orientada a objetos para simulación distribuida en verilog.....	12
Alchemi: Un sistema Empresarial Grid Computing basado en .NET.....	13
Un modelo de flujo de datos para el sistema Grid Computing basado en .NET.....	14
Aneka: La siguiente generación de plataformas empresariales para aplicaciones e-ciencia y e-negocios .....	15
Acelerando la investigación médica usando el veloz sistema de flujo de datos .....	17
2.3 Laboratorios virtuales ventajas .....	18
2.4 Laboratorios virtuales desventajas .....	19





2.5 Tipos de servicios que ofrece un laboratorio virtual .....	20
2.6 Computación en malla.....	21
2.6.1 Concepto de computación en malla.....	21
2.6.2 Ventajas que ofrece la computación en malla .....	21
2.6.3 Introducción a la computación en malla .....	22
2.6.4 Arquitectura de la computación en malla .....	22
2.6.5 Lo que la computación en malla puede hacer .....	24
2.6.5.1 Aprovechando los recursos que no siempre se usan .....	24
2.6.5.2 Capacidad de CPU paralela.....	25
2.6.5.3 El acceso a los recursos adicionales.....	25
2.6.5.4 Balanceo de recursos.....	25
2.7 .NET framework Remoting .....	26
2.7.1 Introducción .....	26
2.7.2 Arquitectura de .NET framework Remoting.....	27
2.7.2.1 Copias y referencias.....	27
2.7.2.2 Arquitectura simplificada de interacción remota.....	28
2.7.2.3 Diseño completo de un sistema de interacción remota .....	28
2.7.3 Convertir objetos en objetos utilizables en forma remota.....	29
2.7.3.1 Procesos y dominios de aplicación .....	29
2.7.3.2 Procesos.....	29
2.7.3.3 Dominios de aplicación .....	30
2.7.4 Periodos y activación de objetos remotos .....	30
2.7.4.1 Activación de objetos remotos .....	30
2.7.4.2 Periodos.....	31
2.7.5 Canales.....	32
<b>CAPITULO 3. DESCRIPCION DE LA ARQUITECTURA DEL SISTEMA .....</b>	<b>33</b>
3.1 Descripción del marco de trabajo.....	34
3.2 Arquitectura del sistema.....	35
3.3 Sistema para la creación de la plataforma estática.....	36
3.3.1 Archivo de Configuración.....	37
3.3.2 Creación de la plataforma estática del sistema.....	38



3.3.3 Contenedor de procesos .....	38
3.4 Maestro .....	39
3.4.1 Componente asociación .....	40
3.4.2 Componente grafo del flujo de datos .....	40
3.4.3 Contenedor unitario de proceso .....	41
3.4.4 Componente planificador .....	41
3.5 Trabajador .....	42
3.5.1 Componente ejecutor .....	43
3.5.2 Sistema de colas .....	44
<b>CAPITULO 4. PRUEBAS Y RESULTADOS.....</b>	<b>46</b>
4.1 Presentación de resultados .....	47
4.1.1 Carga del archivo de configuración de la aplicación y creación del contenedor de procesos .....	47
4.1.2 Creación de la cola de resultados .....	49
4.1.3 Envío del contenedor de procesos al maestro .....	49
4.1.4 Planificación, distribución de procesos y balanceo de las cargas de trabajo.....	50
4.1.5 Obtención de la lista de trabajadores disponibles (componente asociación) .....	51
4.1.6 Creación del grafo de flujo de datos (componente grafo del flujo de datos) .....	51
4.1.7 Envío de los procesos a los trabajadores (componente planificador) .....	53
4.1.8 Ejecución de los procesos (componente ejecutor).....	55
4.1.9 Solucionando la dependencia de datos .....	57
4.1.10 Presentación de resultados al usuario .....	57
4.1.11 Presentación de la bitácora al usuario .....	58
4.2 Presentación de pruebas que evalúan el rendimiento del sistema .....	59
4.2.1 Características de los equipos de cómputo que conforman la grid.....	59
4.2.2 Descripción del desarrollo de la aplicación (sumador de 1 bit con acarreo) utilizada en las pruebas.....	60
Descripción de la arquitectura.....	60
Código en SystemC.....	61
Pre simulación.....	62
Grafo del sistema .....	64
Particionamiento .....	66



<b>CAPITULO 5. CONCLUSIONES Y TRABAJOS A FUTURO.....</b>	<b>70</b>
5.1 Conclusiones.....	71
5.2 Trabajos a futuro .....	72
<b>REFERENCIAS BIBLIOGRAFICAS .....</b>	<b>73</b>

## INDICE DE FIGURAS

Figura 2.1 Arquitectura del DVS.....	13
Figura 2.2 Arquitectura de Alchemi.....	14
Figura 2.3 Arquitectura del modelo de flujo de datos para sistemas Grid Computing basados en .NET .....	15
Figura 2.4 Arquitectura de Aneka.....	17
Figura 2.5 Arquitectura del sistema para acelerar la investigación médica basado en el flujo de datos.....	18
Figura 2.6 Representación de una Grid Computing.....	21
Figura 2.7 Arquitectura de la Grid Computing.....	23
Figura 2.8 Balanceo de recursos .....	26
Figura 2.9 Elementos que componen el framework de .NET Remoting.....	27
Figura 2.10 Arquitectura del framework de .NET Remoting .....	27
Figura 2.11 Proceso de interacción remota .....	29
Figura 3.1 Entornos y herramientas que ofrecen los laboratorios virtuales .....	34
Figura 3.2 Integración del sistema Grid Computing a los laboratorios virtuales.....	35
Figura 3.3 Arquitectura del sistema .....	36
Figura 3.4 Estructura del archivo de configuración.....	37
Figura 3.5 Contenedor de procesos.....	38
Figura 3.6 Estructura del contenedor unitario de proceso .....	41
Figura 3.7 Grafo que describe la comunicación entre procesos de una aplicación.....	44
Figura 4.1 Comparación entre el contenido del archivo de configuración y el contenido contenedor de vértices.....	48



Figura 4.2 Cola de resultados perteneciente a la simulación del sumador de 1 bit con acarreo, a la cual se le asigno el PID 23 .....	49
Figura 4.3 Media aritmética obtenida por los pesos de 8 procesos pertenecientes a la simulación del sumador de 1 bit con acarreo.....	50
Figura 4.4 Características de los Trabajadores disponibles pertenecientes a la Grid.....	51
Figura 4.5 Clasificación de Trabajadores.....	52
Figura 4.6 Asignación de procesos a trabajadores.....	53
Figura 4.7 Creación y envío de los contenedores unitarios de proceso a los trabajadores.....	54
Figura 4.8 Almacenamiento de los procesos en el trabajador <i>192.168.1.73</i> .....	55
Figura 4.9 Almacenamiento de los procesos en el trabajador <i>192.168.1.68</i> .....	56
Figura 4.10 Lectura y escritura en las colas de cada proceso .....	56
Figura 4.11 Agregado de los resultados en la cola de entrada de un proceso.....	57
Figura 4.12 Presentación de resultados al Usuario .....	58
Figura 4.13 Presentación de la bitácora al Usuario.....	58
Figura 4.14 Conversión de código Verilog, VHDL a SystemC.....	62
Figura 4.15 Parsing al código original .....	63
Figura 4.16 Pre simulación para la predicción de las cargas de trabajo en los nodos.....	64
Figura 4.17 Tabla de descripción del grafo .....	65
Figura 4.18 Construcción del grafo en forma grafica .....	65
Figura 4.19 Creación de sub módulos.....	66
Figura 4.20 Compilación de los archivos en C++.....	67
Figura 4.21 Gráfica que muestra el rendimiento del sistema. ....	69

## INDICE DE TABLAS

Tabla 4.1 Características de las computadoras pertenecientes a la Grid.....	47
Tabla 4.2 Características de los equipos de cómputo pertenecientes a la Grid .....	59
Tabla 4.3 Características de la aplicación .....	67
Tabla 4.4 Distribución de los procesos entre los trabajadores.....	68



## GLOSARIO

**Middleware:** Es un software de conectividad que ofrece un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas. Funciona como una capa de abstracción de software distribuida, que se sitúa entre las capas de aplicaciones y las capas inferiores (sistema operativo y red).

**Simulación:** Es la experimentación con un modelo de una hipótesis o un conjunto de hipótesis de trabajo.

**Circuitos VLSI:** Integración en escala muy grande de sistemas de circuitos basados en transistores.

**Balance o balanceo de carga:** Se refiere a la técnica usada para compartir el trabajo a realizar entre varios procesos, ordenadores, discos u otros recursos.

**Proceso:** Es un concepto manejado por el sistema operativo que consiste en el conjunto formado por:

- Las instrucciones de un programa destinadas a ser ejecutadas por el microprocesador.
- Su estado de ejecución en un momento dado, esto es, los valores de los registros de la CPU para dicho programa.
- Su memoria de trabajo, es decir, la memoria que ha reservado y sus contenidos.
- Otra información que permite al sistema operativo su planificación.

**Computación Distribuida:** Es un nuevo modelo para resolver problemas de computación masiva utilizando un gran número de computadoras organizadas en racimos incrustados en una infraestructura de telecomunicaciones distribuida.

**Protocolos:** Son reglas de comunicación que permiten el flujo de información entre computadoras distintas que manejan lenguajes distintos.

**Redes peer to peer:** Son redes que aprovechan, administran y optimizan el uso de banda ancha que acumulan de los demás usuarios en una red por medio de la conectividad entre los mismos usuarios participantes de la red, obteniendo como resultado mucho más rendimiento en las conexiones y transferencias con algunos métodos centralizados convencionales.

**Programación Orientada a Objetos:** Es un paradigma de programación que usa objetos y sus interacciones para diseñar aplicaciones y programas de computadora. Está basado en varias técnicas, incluyendo herencia, modularidad, polimorfismo y encapsulamiento.

**Objeto:** Se define como la unidad que en tiempo de ejecución realiza las tareas de un programa.



---

**Modelado UML:** El Lenguaje de Modelado Unificado (UML:Unified Modeling Language), es la sucesión de una serie de métodos de análisis y diseño orientadas a objetos que aparecen a fines de los 80's y principios de los 90s.UML es llamado un lenguaje de modelado.

**Escalabilidad:** En telecomunicaciones y en ingeniería informática, es la propiedad deseable de un sistema, una red o un proceso, que indica su habilidad para, o bien manejar el crecimiento continuo de trabajo de manera fluida, o bien para estar preparado para hacerse más grande sin perder calidad en los servicios ofrecidos.

**Script:** Es un guión o conjunto de instrucciones. Permiten la automatización de tareas creando pequeñas utilidades. Es muy utilizado para la administración de sistemas UNIX. Son ejecutados por un intérprete de línea de órdenes y usualmente son archivos de texto. También un script puede considerarse una alteración o acción a una determinada plataforma.



# Capítulo 1

## *Introducción*

---

**E**n este capítulo se presenta la descripción del proyecto de investigación, donde se muestra el problema que se pretende resolver; así como los objetivos, solución y metodología propuesta para enfrentar dicho problema.

A continuación se describe el contenido de este capítulo:

- Se describe a la educación virtual y los laboratorios virtuales.
- Posteriormente, se describe el planteamiento del problema.
- Se presenta también la justificación y los objetivos alcanzados para lograr una solución al problema.
- Se menciona el alcance del proyecto y las limitaciones que éste presenta.
- Por último, se presenta la estructura que conforma la tesis.





### 1.1.PLATAFORMAS VIRTUALES

Son escenarios educativos diseñados de acuerdo a una metodología de acompañamiento a distancia. También se pueden ver como herramientas basadas en páginas web para la organización e implementación de cursos, talleres, laboratorios o para apoyar actividades educativas presenciales.

Este tipo de plataformas se están volviendo un instrumento en forma cada vez más utilizado en ámbitos de enseñanza. Este hecho no solo está relacionado con factores puramente tecnológicos sino también se debe a que gracias a estas plataformas va en aumento la oferta de programas informáticos gracias al vertiginoso desarrollo de la Web; aunado a esto, se deriva la urgente necesidad que tienen los centros académicos de ampliar su oferta educativa, adaptarla a las necesidades de la sociedad y hacerla accesible a un mayor número de estudiantes.

### 1.2.LA EDUCACIÓN VIRTUAL: ¿UNA ALTERNATIVA A LA EDUCACIÓN TRADICIONAL?

La posibilidad de transmitir conocimientos, imágenes, textos, sonido, recomendaciones, por Internet al hogar, y a una multitud de usuarios simultáneamente a un costo más o menos reducido, y en el futuro a un costo relativamente mínimo, está revolucionando la enseñanza tradicional en sus dos modalidades presencial y a distancia y en medio de éstas dos surge una tercera y nueva modalidad de enseñanza: la enseñanza virtual o enseñanza online. Se trata de una forma de enseñanza basada en las nuevas tecnologías de la información y la comunicación fundamentalmente en Internet que promete "revolucionar" la educación tradicional.

En este sentido es interesante analizar si la educación virtual puede sustituir las modalidades tradicionales y por lo tanto saber si presenta o no nuevas y concretas posibilidades de elección para el educando.

Ésta es precisamente la lógica subyacente al desarrollo de propuestas educativas en Internet: las nuevas tecnologías presentan a priori una posibilidad de elección entre la educación presencial y la educación virtual.

En este sentido, a la pregunta: "una nueva opción: ¿para quién?", la educación a distancia a través de Internet responde:



## Capítulo 1 Introducción



1. Educación virtual para personas sin acceso al sistema educativo tradicional (por ejemplo, personas aisladas geográficamente)
2. Educación virtual para personas con acceso al sistema educativo tradicional.

Ahora bien, una vez definidos los potenciales grupos destinatarios de la educación virtual, deberíamos preguntarnos si la educación virtual es al menos tan efectiva como la educación tradicional.

Claro que esta pregunta no es tan significativa en el primer grupo de destinatarios (personas sin acceso al sistema educativo tradicional) dado que no se trata de una nueva opción (entendiendo *opción* como *alternativa*), sino de una oportunidad sin precedentes.

Gracias a Internet, los estudiantes no tienen que ir a la escuela, y pueden tener su propio ritmo de estudio de acuerdo a los horarios que más les convenga (aprendizaje asíncrono) lo cual reduce el costo de oportunidad de la educación y permite insertar dentro del mercado educativo a alumnos que nunca antes hubieran podido formarse dentro del sistema educativo tradicional.

De manera que mientras la educación virtual arroje algún resultado positivo (superior a cero) en los indicadores de aprendizaje, debería ser un gran logro, además de que se tienen que recibir con alegría las nuevas oportunidades que las tecnologías de la comunicación y de la información ofrecen para aquellas personas que anteriormente no tenían acceso a ninguna propuesta educativa.

Ahora bien, ¿qué sucede con el segundo grupo de análisis? (personas con acceso al sistema educativo tradicional).

Aquí el análisis de efectividad se vuelve más interesante dado que para promover la elección entre programas de educación virtual y los programas tradicionales debería poder asegurar a ciencia cierta que los primeros son al menos tan efectivos como los segundos.

Y la realidad es que a la pregunta: ¿Qué tan efectiva es la educación vía Internet?

La respuesta es: "TODAVÍA NO SE SABE".

En los últimos años se observa en el mundo una tendencia importante hacia la educación con fines de lucro (la privatización de la educación) y un aumento considerable de la demanda de educación a



distancia. Austan Goolsbee de la Universidad de Chicago, analiza estas dos tendencias y demuestra que si bien hubo un crecimiento rápido de la Internet educativa, su impacto en la productividad educativa no será demasiado grande en los próximos años.

Lo que nos ha llevado a pensar... ¿Cómo podemos apoyar de una manera más significativa a todas aquellas personas que tienen o no acceso a la educación tradicional?, en concreto a personas que les interese tener una herramienta que les permita realizar una mayor cantidad de interacciones (por ejemplo: pruebas o simulaciones a cualquier diseño que estén desarrollando en laboratorios de carreras que formen parte del área de las ciencias, ingeniería y el comercio), ya que de esta manera obtendrán una percepción de que su interacción con un sistema informático es equivalente al que obtendrían en un esquema presencial tradicional.

### 1.3.LABORATORIOS VIRTUALES

Se puede definir como un sistema informático que permite, operar dentro o fuera de línea, modelar interactivamente lugares, objetos o situaciones con fines de experimentación, investigación y/o observación, contribuyendo a incrementar la capacidad de metodología de búsqueda por el participante así como de diseño experimental.

Trasladando este entorno a la enseñanza actual, los elementos necesarios para abordar la realización de actividades prácticas son los laboratorios virtuales (LV) y remotos (LR), accesibles a través de Intranet, Internet o ambientes computacionales, donde el alumno realiza las practicas de una forma lo más similar posible a como si estuviese en las dependencias del laboratorio tradicional (LT), simulando e interactuando con instrumentos virtuales.

En el laboratorio tradicional (LT), los recursos en personas y espacios son restringidos, debido a su masificación y a problemas presupuestarios; se requiere la presencia física del estudiante y la supervisión del profesor. Una solución a estos problemas la encontramos en la aplicación de los avances tecnológicos a la docencia e investigación universitaria y, en concreto, el uso de laboratorios virtuales (LV) y remotos (LR). El LV acerca y facilita la realización de experiencias a un mayor número de alumnos, aunque alumno y laboratorio no coincidan en el espacio. Permite simular fenómenos y modelos físicos, conceptos abstractos, mundos hipotéticos, controlar la escala de tiempo, etc, ocultando el modelo matemático y mostrando el fenómeno simulado de forma



interactiva. La creciente complejidad de las actividades en el LT y el desarrollo de las Tecnologías de Comunicación e Información y la Computación, han hecho que los LV evolucionen, transformándose en laboratorios remotos (LR), donde el alumno utiliza y controla los recursos del laboratorio, a través de una red local (Intranet) o bien a través de Internet.

### **1.4. USO DE LABORATORIOS VIRTUALES EN ÁREAS DE LA CIENCIAS**

Todas aquellas áreas que pertenecen a las ciencias, ingeniería y al comercio están adoptando como un recurso informático justificado a los laboratorios virtuales, por un razonamiento primordial: el reemplazar equipamiento real, de altísimo costo, por simulaciones de equipamiento conducidas y compartidas a distancia, en modalidad sincrónica o asincrónica, en el medio virtual, por la gran cantidad de datos que necesitan analizar y procesar, por el alto grado de interacción que ofrecen, por brindar una mayor capacidad de modelación (gráficas tridimensionales).

Además se está volviendo cada día más necesario poder tener en los laboratorios virtuales una forma de resolver tareas intensivas, las cuales demandan una gran cantidad de recursos computacionales, como podrían ser, el cálculo de ecuaciones matemáticas complejas, el análisis de fenómenos físicos. Por otra parte, existen tanto empresas como estudiantes de ingeniería que crean diseños de productos que posteriormente pueden vender, por lo cual es indispensable para ellos estar completamente seguros de que sus diseños han sido completamente probados en forma exhaustiva antes de lanzarlos al mercado.

Para lo cual se ha vislumbrado como una alternativa el poder ofrecerles a todas estas personas, tener a su disposición a través de Internet, laboratorios virtuales que tengan la capacidad de realizar la ejecución de aplicaciones intensivas en forma distribuida.

### **1.5. PLANTEAMIENTO DEL PROBLEMA**

Para cubrir la necesidad creciente de demandas en los laboratorios virtuales, en cuanto a poder hacer frente a las tareas que son muy demandantes computacionalmente para ser resueltas por una sola computadora aislada, es necesario hacer uso de sistemas de computadoras distribuidos y paralelos [3].



Se plantea para este proyecto de investigación, crear un sistema capaz de realizar cómputo distribuido para disminuir el tiempo de procesamiento de las aplicaciones que requieren resolver tareas intensivas.

Donde se tiene como un caso particular, simplificar la tarea a todas aquellas personas que necesiten realizar simulaciones muy demandantes en cuanto a cómputo de sus aplicaciones; ya que ellos requieren de un sistema que les permita una ejecución en forma distribuida, para poder realizar una mayor cantidad de tareas en menor tiempo.

### 1.6.JUSTIFICACIÓN

Las Grids computacionalmente acoplan recursos geográficamente distribuidos, por tanto se está volviendo una computación efectiva para resolver problemas de gran escala en la ciencia, ingeniería y en el comercio [4]. Dado que la computación y la tecnología de comunicación han tenido un impacto significativo en los sistemas de educación. Esta tecnología ha mejorado el aprendizaje colaborativo en línea, además de eso mejora las experiencias de aprendizaje de los usuarios [5]. El crear laboratorios virtuales para la simulación de aplicaciones en forma distribuida ayudará a ofrecer a los estudiantes de ingeniería y a las empresas que se dedican a vender sus productos, una herramienta con la cual podrán realizar simulaciones de sus diseños en tiempos considerablemente menores a los que ahora tienen, logrando con ello lanzar al mercado sus productos más rápidamente, lo que les podría permitir generar mayores ganancias. También puede lograr reducir los problemas de extenuación de recursos que se llegan a presentar en las simulaciones que requieren alto poder computacional.

Por tanto es importante que estos laboratorios puedan realizar simulación distribuida, ya que el desarrollo de aplicaciones modernas se está volviendo progresivamente complicado planteando un reto interminable para la simulación secuencial. Para acomodar la necesidad de disminución del tiempo de simulación, está siendo bastante necesario el uso de simulación distribuida [24].

Día con día está surgiendo la necesidad de trabajar sobre ambientes de diseño electrónico basados en red, por lo cual es necesario que estos ambientes tengan la capacidad de ser escalables, adaptativos, seguros, con alta disponibilidad y bajos costos [5], para que de esta manera se puedan ofrecer mejores servicios de simulación.



Por tanto, es en este punto, la importancia que tiene desarrollar sistemas de laboratorios virtuales que aprovechen la tecnología Grid Computing, ya que ésta trabaja bajo el esquema de reunir sistemas computacionales heterogéneos no importando su ubicación geográfica bajo una sola malla, para que de esa manera se puedan compartir recursos computacionales y así poder ofrecer reducción en los tiempos de simulación.

### 1.7.OBJETIVOS

#### 1.7.1. OBJETIVO GENERAL

Desarrollar un sistema para la ejecución distribuida de procesos en una Grid integrada por computadoras de escritorio (Desktop Computing) heterogénea, utilizando P2P como protocolo de comunicación en el paso de mensajes entre procesos.

#### 1.7.2. OBJETIVOS PARTICULARES

- Desarrollar un aplicación que cree/genera un ambiente estático de una cierta aplicación a ejecutar en el contexto distribuido.
- Desarrollar un modelo para el balance de carga en las unidades de ejecución, tomando en cuenta los requerimientos de cada proceso y recursos disponibles en cada unidad de ejecución.
- Desarrollar un sistema para la distribución de procesos utilizando el modelo de balance de carga.
- Desarrollar un modelo para el paso de mensajes entre procesos utilizando peer to peer.
- Desarrollar un sistema de colas para el paso de mensajes entre procesos.
- Implementar un sistema que ocupe el modelo de pasos de mensajes y que inicie la ejecución de los procesos en las unidades de ejecución.



### 1.8. ALCANCES

El diseño del sistema, que es el punto central en esta investigación, permitirá realizar simulaciones, reduciendo el tiempo de la simulación gracias a que se realizará en forma distribuida sobre una Grid Desktop Computing, creada por medio de la tecnología .NET Remoting. Además de ello, tendrá la capacidad de ofrecer el soporte de comunicación peer to peer para múltiples procesos que tienen dependencias de datos. Los Usuarios no necesitan preocuparse acerca de los detalles de los procesos, hilos y las comunicaciones explícitas.

La distribución, planificación y balanceo de las cargas de trabajo se realizarán de manera automática y sin la intervención o supervisión de un administrador de sistemas. La presentación de resultados y la generación de la bitácora serán presentados por el sistema al o los usuarios. Además de ello, el sistema en global tendrá la gran ventaja de que puede trabajar con computadoras heterogéneas que formen parte de la Grid.

### 1.9. LIMITACIONES

No sola la planificación, distribución y balanceo de las cargas de trabajo conforman una solución integral para la reducción en los tiempos de simulación y en la prevención de la extenuación de recursos; ya que primero se tiene que tener en cuenta que una de las principales bases de este sistema es que tiene que recibir un conjunto de procesos previamente particionados, de tal manera que dichos procesos se logren poder ejecutar paralelamente en forma eficaz.

Se tiene como una limitación que el sistema actualmente ha sido puesto a prueba con simulaciones de circuitos VLSI que no permiten determinar de manera cuantitativa la capacidad que se espera. Sin embargo, la implementación realizada asegura que dichas pruebas serán soportadas.

Otra limitación que se encuentra, es que en este momento el sistema no sabe cuando ha finalizado una simulación, por tanto para poder finalizarla, se debe de hacer de forma manual cerrando los procesos que se encuentren abiertos.





Además, la cantidad de cómputo en cada simulación debe ser lo bastante significativa como para poder observar la ventaja que se tiene al realizar simulaciones distribuidas en contra de simulaciones centralizadas.

Para que la computación distribuida genere beneficios se tiene que contar con un número significativo de computadoras con ciertas características en cuanto al poder de cómputo.

### **1.10. ESTRUCTURA DE LA TESIS**

CAPITULO 1.- Presenta la introducción de esta investigación, donde se muestra el planteamiento del problema, la justificación, objetivos, alcances y limitaciones.

CAPITULO2.- Proporciona el estado del arte en los sistemas que ofrecen laboratorios virtuales para la simulación, y en general sobre los sistemas Grid Computing y sus usos, además de mostrar sus conceptos básicos y los de la tecnología .NET Remoting, la cual se implementó para generar la Grid en este proyecto de investigación

CAPITULO 3.- Describe la arquitectura del sistema y la interacción que hay entre sus elementos.

CAPITULO 4.- Muestra los resultados obtenidos además de que se especifican las pruebas realizadas.

CAPITULO 5.- Se presentan las conclusiones y trabajos a futuro.



# Capítulo 2

## *Marco Teórico*

---

**E**n este capítulo se da una descripción de los laboratorios virtuales así como de los trabajos relacionados, de los cuales se han tomado ideas para el desarrollo de este proyecto, finalizando con la descripción de las herramientas y tecnologías, ocupadas para el desarrollo de este proyecto de investigación.

El contenido de los temas presentados en este capítulo incluye:

- Historia del Grid Computing.
- Presentación de trabajos relacionados con el Grid Computing.
- Características de los laboratorios virtuales.
- Descripción, origen y evolución de los sistemas Grid Computing.
- Descripción de las características que conforman la tecnología .NET Remoting.



### 2.1. HISTORIA DE LA COMPUTACIÓN EN MALLA

La palabra Grid en inglés significa *malla* y hace referencia a la red eléctrica. Para obtener electricidad, simplemente debe haber conexión a cualquier punto de la *malla eléctrica (power grid)* sin preocuparse por las plantas generadoras que interconectadas brindan esta omnipresencia del servicio; es decir, no se sabe de dónde proviene la electricidad que se utiliza, simplemente se aprovecha.

Este fue el concepto que adoptaron Carl Kesselman e Ian Foster en su libro *The Grid: Blueprint for a new computing infrastructure*, publicado en 1998.

La idea del Grid Computing (que incluye cómputo distribuido, programación orientada a objetos y Servicios Web) fue traída por Ian Foster, Carl Kesselman, y Steve Tueque, ampliamente apreciados como los padres del Grid. Ellos condujeron el esfuerzo para crear el Globus Toolkit. No sólo incorporando administración en la computación, sino también administración en el almacenamiento, proporcionando seguridad, movimiento de datos, monitoreo y un kit de herramientas para desarrollar servicios adicionales basados en la misma infraestructura [6].

### 2.2. PRESENTACIÓN DE TRABAJOS RELACIONADOS CON ESTE PROYECTO DE INVESTIGACIÓN

La programación de flujo de datos es natural y conveniente para describir sistemas de procesamiento de señales digitales, pero en tiempo de ejecución tienen un gran costo. En algunas situaciones los diseñadores no están dispuestos a gastar dinero en recursos de cómputo. Esto es particularmente verdadero cuando la máquina es un circuito integrado programable, tal y como un DSP (Digital Signal Processing, Procesamiento digital de señal). Sin embargo, en tiempo de ejecución la sobrecarga en las implementaciones LGDF (Flujo de Datos de Grano Largo, Large Grain Data Flow) no requieren más sistemas de procesamiento de señales, ya que cada sistema es en su mayor parte asíncrono. El SDF (Synchronous Data Flow, Flujo de Datos Síncronos) difiere del tradicional flujo de datos en que la cantidad de datos producidos por un nodo de flujo de datos se especifica antes para cada entrada y salida. Esto es equivalente a especificar la prueba relativa evaluada por el sistema de procesamiento de señales. Esto significa que la planificación de un nodo SDF no necesita terminar en tiempo



de ejecución, porque puede terminar en tiempo de compilación (estáticamente), de esta manera se evaporan las sobrecargas en tiempo de ejecución. La prueba evalúa todas las posibles diferencias, las cuales no son verdaderas en los más frecuentes manejadores de datos; en las metodologías de programación para el procesamiento de señales. El flujo de datos síncrono está estrechamente relacionado con los grafos computacionales, un caso especial de redes Petri.

Uno de los trabajos que se relacionan con este proyecto de investigación fue presentado en 1987, donde se desarrolla la teoría necesaria para planificar estáticamente programas SDF en individuales o múltiples procesadores [11]. Una clase estática de algoritmos de planificación es provista válidamente y los algoritmos específicos son dados para la planificación de sistemas SDF en simples o múltiples procesadores.

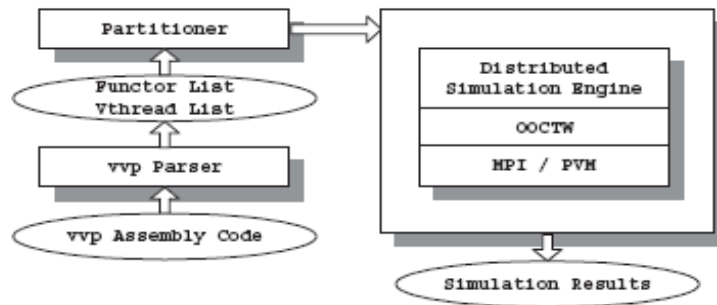
Hay un amplio uso de los lenguajes de descripción de hardware (HDL) para acelerar el tiempo de comercialización utilizado en el diseño de los sistemas digitales modernos. Para la verificación, los ingenieros pueden simular el hardware con el fin de comprobar su rendimiento y la corrección con la ayuda de un HDL. Sin embargo, la simulación no se puede mantener al ritmo con el crecimiento en tamaño y complejidad de los circuitos y se ha convertido en un cuello de botella del proceso de diseño. La simulación distribuida sobre el HDL en un cluster de supercomputadoras tiene el potencial de proporcionar una solución a este problema.

Otro trabajo relacionado con este trabajo de investigación es el que se presentó en el año del 2003, el cual muestra el diseño y la implementación de un DVS (un área de trabajo orientada a objetos para la distribución de simulaciones Verilog). Verilog es un HDL usado en la industria. DVS es el resultado del clúster Time Warp, originalmente desarrollado para la simulación lógica. El diseño del área de trabajo hace énfasis en la simplicidad, extensibilidad y facilidad para envolver experimentos particionados y el balanceo dinámico de cargas de trabajo. (Véase Figura 2.1).

Verilog y VHDL son importantes lenguajes de diseño VLSI. Sin embargo, los esfuerzos de investigación se han centrado hasta el momento en simuladores distribuidos VHDL. Este artículo presenta una descripción de esta investigación, la cual describe el área de trabajo para la simulación distribuida en Verilog.



Escribir un compilador de Verilog representa un importante compromiso. Los autores hicieron uso de Icarus Verilog, un compilador Verilog de código abierto que además es un simulador. También se dieron a la tarea de rediseñar el Time Warp, y se ha utilizado éste como backend para el entorno de simulación [3].



**Figura 2.1.** Arquitectura del DVS.

Las Grids computacionalmente acoplan recursos geográficamente distribuidos, por tanto se está volviendo una computación efectiva para resolver problemas de gran escala en la ciencia, la ingeniería y en el comercio. El software para habilitar el Grid Computing ha sido primordialmente escrito para sistemas operativos Unix, así gravemente limitan la habilidad para utilizar efectivamente los recursos computacionales de las computadoras de escritorio basadas en Windows. La Grid Computing basada en Windows es particularmente importante desde el punto de vista de las industrias.

Otros trabajos, que si bien no se relacionan con la parte de la simulación de los circuitos digitales, si tienen mucho que ver con el modelo de flujo de datos en las Grid, el cual ha sido implementado en el sistema descrito en esta tesis. A continuación se mencionan algunos de ellos:

Alchemi, un sistema empresarial basado en .NET que provee una maquinaria en tiempo de ejecución y un ambiente de programación para construir Grids computadoras/escritorio y desarrollar aplicaciones Grids. Esto permite la composición flexible de aplicaciones soportando un modelo orientado a objetos para programar aplicaciones. El soporte de la plataforma es provisto por medio de una interfaz de servicios Web y un modelo flexible para la ejecución en nodos Grid dedicados o no dedicados. [9]. Este trabajo data del año 2005.

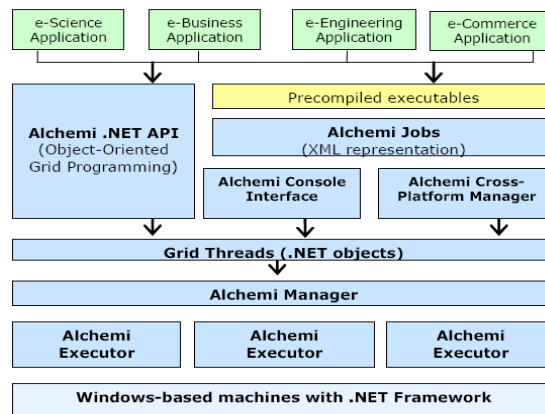


## CAPITULO 2 Marco Teórico



Alchemi fue concebido con el objetivo de hacer la construcción Grid y el desarrollo de software Grid tan fácil como sea posible sin sacrificar la flexibilidad, la adaptabilidad, la fiabilidad y la extensibilidad. (Véase Figura 2.2). Las características que soporta Alchemi son:

- Un clustering basado en Internet de computadoras de escritorio heterogéneas.
- Nodos individuales para la ejecución, dedicados o no dedicados (voluntarios).
- Un modelo de programación orientado a objetos para aplicaciones Grid.
- Un modelo basado en archivos.
- Una interfaz para los servicios web que soportan el modelo de trabajo para la interoperabilidad con el middleware.



**Figura 2.2.** Arquitectura de Alchemi.

El trabajo que más incidencia tiene hacia este proyecto es el que se presentó en el año 2007, el cual fue llamado “Un modelo de flujo de datos para sistemas Grid Computing basados en .NET”, está encaminado hacia el soporte avanzado de aplicaciones que contienen múltiples tareas con relaciones entre la dependencia de datos. Algunas aplicaciones de recursos intensivos consisten de múltiples módulos, los cuales reciben entradas de datos, realizan operaciones y generan resultados. Aplicaciones científicas para esta naturaleza incluyen: simulaciones, minería de datos, y grafos de computación; en muchos casos para esas aplicaciones un módulo de salida de datos se convierte en la entrada de otros módulos. El modelo “coarse grained dataflow” puede usarse para describir tales aplicaciones.

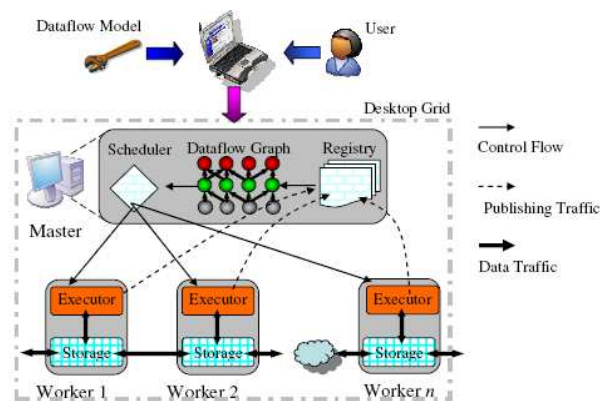


De esta manera ellos usan el modelo de programación de flujo de datos, para crear un grafo de flujo de datos, donde especifican la relación entre la dependencia de datos dentro de aplicaciones distribuidas. Debajo de la interfaz del flujo de datos, usan el motor de flujo de datos para explorar el grafo para planificar tareas de acuerdo a los recursos distribuidos y el manejo de los problemas difíciles, como ejecuciones escalables, tolerancia a fallos y balanceo de cargas. Dentro de este proceso los usuarios no necesitan preocuparse acerca de los detalles de los procesos, hilos y las comunicaciones explícitas.

Las contribuciones principales de este trabajo son:

- Un simple y poderoso modelo de programación de flujo de datos, el cual soporta la composición de aplicaciones paralelas para desarrollar en ambientes distribuidos
- Una arquitectura y maquinaria en tiempo de ejecución que soporte la planeación de la computación de flujo de datos en ambientes dinámicos y el manejo transparente de fallos.
- Un análisis detallado del modelo de flujo de datos usando dos aplicaciones de ejemplo sobre computadoras de escritorio Grid [7].

En la Figura 2.3 se muestra la arquitectura del sistema descrito anteriormente.



**Figura 2.3.** Arquitectura del modelo de flujo de datos para sistemas Grid Computing basados en .NET.

También en el 2007 fue presentado el diseño de Aneka, una plataforma .NET desktop Grid Computing orientada a servicios la cual provee: 1) un contenedor de servicio configurable que almacena servicios para descubrir, planificar y balancear cargas de trabajo y 2) un framework



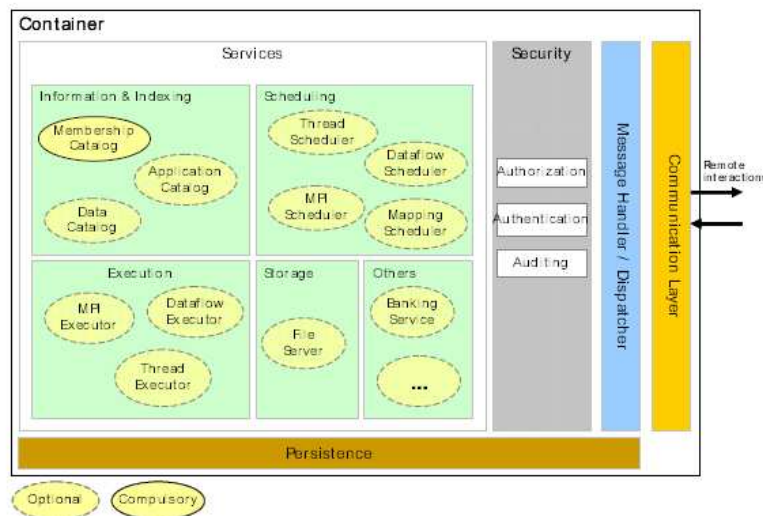


/API para soportar varios modelos de programación en los que se incluyen threading, batch processing, MPI y dataflow. Los usuarios y desarrolladores pueden fácilmente programar diferentes modelos y proveer los servicios del contenedor para correr sus aplicaciones sobre Desktop Grids manejados por Aneka. De esta manera se presenta la implementación de ambos tanto lo esencial como los servicios avanzados que contiene la plataforma. Evaluando de esta manera el sistema con aplicaciones usando los modelos Grid task y dataflow [8].

Aneka fue creada con el fin de proveer un conjunto de servicios que realicen la construcción Grid y el desarrollo de aplicaciones tan fácil como fuera posible sin sacrificar flexibilidad, escalabilidad, fiabilidad y extensibilidad. (Véase Figura 2.4).

Las mejores características de Aneka son:

- Un contenedor configurable habilitador de servicios, soluciones persistentes, implementaciones de seguridad y protocolos de comunicación.
- Una arquitectura descentralizada permitiendo nodos individuales.
- Múltiples modelos de programación incluyendo el orientado a objetos, Grid threading, el modelo de programación (fine-grained abstraction), el modelo file-based Grid task (coarse-grained abstraction) y el modelo dataflow.
- Múltiples mecanismos de autenticación como seguridad basada en roles, certificados X. 509/GSI proxys y autenticaciones de dominios de Windows.
- Múltiples opciones de persistencia de datos como son RDBMS, ODBMS y archivos XML.
- Interfaces de servicios web que soportan el modelo de tareas para interoperabilidad con algún intermediario.



**Figura 2.4.** Arquitectura de Aneka.

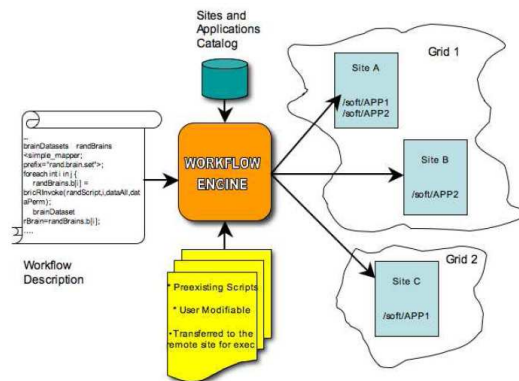
Otro sistema que se ocupa de las tareas anteriormente mencionadas y que data de 2007, utiliza un simple lenguaje de scripts, SwiftScript, tiene provista la especificación de alto nivel de flujo de trabajo que invoca varias aplicaciones programadas sobre grandes cantidades de trabajo potencialmente, el motor del Swift está hecho para la ejecución eficiente de esos flujos de trabajo sobre computadoras secuenciales, computadoras paralelas, y/o Grids distribuidas que obtiene los recursos computacionales de varios sitios. No menos importante, el catalogo de procedencia Swift sigue la pista de todas las acciones realizadas, ocupándose de funciones vitales de contabilidad que tantas veces son causa de dificultades en la computación a gran escala [10]. (Véase Figura 2.5).

Describen su éxito en aplicaciones basadas en imágenes de investigación neurológica, en la cual se trata de expandir el alcance y la escala de sus capacidades de cómputo para el estudio de los mecanismos neurológicos del proceso de recuperación de afasia debido al golpe. Los beneficios de usar el sistema de flujo de trabajo Swift para esta aplicación incluyen grandes reducciones en los datos y en los recursos computacionales.

Esto es típicamente requerido en la investigación científica moderna. Este es almacenado automáticamente, en otro caso es una ardua labor manual, además de proveer un acceso transparente de demanda sobre los recursos de la Grid, nuestro flujo de trabajo también exhibe reproducibilidad y procedencia en el rastreo de los datos resultado, eso habilita la colaboración



en el proceso de investigación actual, no solo en compartir los resultados, sino en el uso compartido y el aprovechamiento de los procesos de investigación actuales.



**Figura 2.5.** Arquitectura del sistema para acelerar la investigación médica basado en el flujo de datos.

### 2.3.LABORATORIOS VIRTUALES: VENTAJAS

Un laboratorio virtual (LV) es un sistema computacional que pretende aproximar el ambiente de un Laboratorio Tradicional (LT). Los experimentos se realizan paso a paso, siguiendo un procedimiento similar al de un LT: se visualizan instrumentos y fenómenos mediante objetos dinámicos (applets de Java o Flash, cgi-bin, javascripts,...), imágenes o animaciones. A continuación, se destacan algunas ventajas importantes de los LV [2]:

- Acerca y facilita a un mayor número de alumnos la realización de experiencias, aunque alumno y laboratorio no coincidan en el espacio. El estudiante accede a los equipos del laboratorio a través de un navegador, pudiendo experimentar sin riesgo alguno, y, además, se flexibiliza el horario de prácticas y evita la saturación por el traslape con otras actividades.
- Reducen el costo del montaje y mantenimiento de los LT, siendo una alternativa barata y eficiente, donde las personas simulan los fenómenos a estudiar como si los observara en el LT.
- Es una herramienta de auto aprendizaje, donde el alumno altera las variables de entrada, configura nuevos experimentos, aprende el manejo de instrumentos, personaliza el experimento, etc. La simulación en el LV, permite obtener una visión más intuitiva de



aquellos fenómenos que en su realización manual no aportan suficiente claridad gráfica. El uso de LV da lugar a cambios fundamentales en el proceso habitual de enseñanza, en el que se suele comenzar por el modelo matemático. La simulación interactiva de forma aislada posee poco valor didáctico, ésta debe ser embebida dentro de un conjunto de elementos multimedia que guíen al alumno eficazmente en el proceso de aprendizaje. Se trata de utilizar la capacidad de procesamiento y cálculo del ordenador, incrementando la diversidad didáctica, como complemento eficaz de las metodologías más convencionales.

- Los estudiantes aprenden mediante prueba y error, sin miedo a sufrir o provocar un accidente, sin avergonzarse de realizar varias veces la misma práctica, ya que pueden repetirlas sin límite; sin temor a dañar alguna herramienta o equipo. Pueden asistir al laboratorio cuando ellos quieran, y elegir las áreas del laboratorio más significativas para realizar prácticas sobre su trabajo.

### **2.4.LABORATORIOS VIRTUALES: DESVENTAJAS**

- El LV no puede sustituir la experiencia práctica altamente enriquecedora del LT. Ha de ser una herramienta complementaria para formar a la persona y obtener un mayor rendimiento.
- En el LV se corre el riesgo de que el alumno se comporte como un espectador. Es importante que las actividades en el LV, vengán acompañadas de un guión que explique el concepto a estudiar, así como las ecuaciones del modelo utilizado. Es necesario que el estudiante realice una actividad ordenada y progresiva, conducente a alcanzar objetivos básicos concretos.
- El alumno no utiliza elementos reales en el LV, lo que provoca una pérdida parcial de la visión de la realidad. Además, no siempre se dispone de la simulación adecuada para el tema que el profesor desea trabajar. En Internet existe demasiada información, a veces inútil. Para que sea útil en el proceso de enseñanza/aprendizaje. Son pocas las experiencias realizadas con LV en los centros educativos, donde aún impera el uso de recursos tradicionales, tanto en la exposición de conocimientos en el aula como en el laboratorio.
- En ocasiones ciertas simulaciones pueden requerir una cantidad de recursos bastante grande ya que realizan tareas bastante intensivas en cuanto a cómputo, como para poder ser ejecutadas en una sola computadora o servidor.



En base a esta última desventaja surge una pregunta: ¿Las simulaciones pueden ser realizadas bajo un esquema de cómputo distribuido?

Todas aquellas simulaciones que puedan ser paralelizables son adecuadas para trabajar bajo el esquema distribuido, donde se tienen varias alternativas para ello:

- **Computación distribuida:** Es una red de ordenadores donde los recursos informáticos son compartidos con todos los otros ordenadores en el sistema. La potencia de procesamiento, la memoria y el almacenamiento de datos, son recursos de la comunidad donde los usuarios autorizados pueden entrar y realizar ciertas tareas.
- **Multi - hilo:** La habilidad de un sistema operativo para ejecutar diferentes partes de un programa, llamados hilos, simultáneamente. El programador debe diseñar cuidadosamente el programa de tal manera que todos los hilos pueden ejecutar al mismo tiempo, sin interferir unos con otros.
- **Modelo de paso de mensajes:** En los modelos de paso de mensajes, los procesos están activos y la información es intercambiada usando en mecanismo de paso de mensajes de una forma u otra. Aunque el mecanismo puede resultar poderoso, fuerza al programador a considerar exactamente donde una comunicación muy costosa debe tener lugar.
- **Grid Computing:** Se enmarca dentro de la tecnología de computación distribuida. Su funcionalidad principal es la de compartir potencia computacional. Se basa en el aprovechamiento de los ciclos de procesamiento no utilizados por los millones de ordenadores conectados a la Red. De esta forma se consigue que puedan resolver de forma distribuida tareas que son demasiado intensivas para ser resueltas por una máquina aislada.

### 2.5. TIPOS DE SERVICIOS QUE OFRECE UN LABORATORIO VIRTUAL

- **Simulación digital.** Servicio que puede ser paralelizable.
- **Simulación analógica.** Servicio que puede ser paralelizable.
- **Compilación.** Servicio no que puede ser paralelizable.
- **Auto routing.** Servicio que puede ser paralelizable.
- **Herramientas.** Osciloscopio, generadores de ondas, gestores de cables, gestores de chips, entre otros. No pueden ser paralelizables.

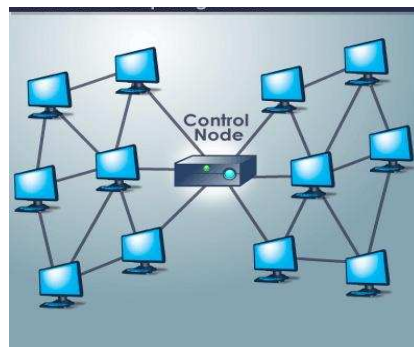


### 2.6.COMPUTACIÓN EN MALLA

#### 2.6.1. CONCEPTO DE COMPUTACIÓN EN MALLA

La funcionalidad principal de la Grid Computing es compartir potencia computacional [12]. Desde el momento en el que los primeros ordenadores comenzaron a conectarse a Internet, surgió la idea de unir la potencia inutilizada de cada uno para abordar problemas a los que sólo podían enfrentarse las supercomputadoras pertenecientes a organizaciones gubernamentales, universidades o grandes multinacionales.

La tecnología que hace esto posible se llama Grid Computing (malla de ordenadores) y se basa en el aprovechamiento de los ciclos de procesamiento no utilizados por los millones de ordenadores conectados a la Red (Véase Figura 2.6). De esta forma se consigue que puedan resolver tareas que son demasiado intensivas para ser resueltas por una máquina aislada.



**Figura 2.6.** Representación de una Grid Computing.

#### 2.6.2. VENTAJAS QUE OFRECE LA COMPUTACIÓN EN MALLA

Los principales beneficios son:

- Ofrecer flexibilidad para llenar las necesidades cambiantes del negocio.
- Brindar alta calidad a menor costo.
- Facilitar el pronto retorno de las inversiones.
- No necesitar de toda una nueva infraestructura para que funcione.
- Facilitar poder de computación / precio barato.



- Brindar el poder de un supercomputador.
- Utilizar software gratuito y usar código fuente abierto.
- No precisar hardware adicional, para posibilitar el incremento de la potencia de cómputo.
- Brindar transparencia para el usuario que participa en el Grid.

### 2.6.3. INTRODUCCIÓN A LA COMPUTACIÓN EN MALLA

La idea del Grid está enfocada fundamentalmente en el acceso remoto a recursos computacionales y pretende ser un paradigma de desarrollo no centrado en una tecnología concreta [13].

La evolución de la Grid Computing se refleja en el avance de la estandarización de esta tecnología (el estándar de Globus Project es el estándar de facto) donde se encuentra definida la arquitectura del Grid, los niveles de acceso, los requisitos, los servicios, etc.

La Grid Computing se enmarca dentro de la tecnología de computación distribuida englobando conceptos como: sistemas operativos distribuidos, programación multiprocesador, computación paralela, redes de computadoras, seguridad, bases de datos, etc. De alguna manera el concepto de Grid Computing da una unidad conceptual a estos problemas de manera que todos ellos puedan verse desde una perspectiva Grid.

La Grid Computing es más que una idea ambiciosa, ya que no sólo se trata de compartir ciclos de CPU para realizar cálculos complejos sino que se busca la creación de una infraestructura distribuida. Esta ardua tarea involucra labores de definición de la arquitectura general, de interconexión de diferentes redes, de definición de estándares, de desarrollo de procedimientos para la construcción de aplicaciones, etc. [14].

La Grid es una idea que promete revolucionar el mundo de la computación y el cómo se desarrollan las aplicaciones actualmente [12].

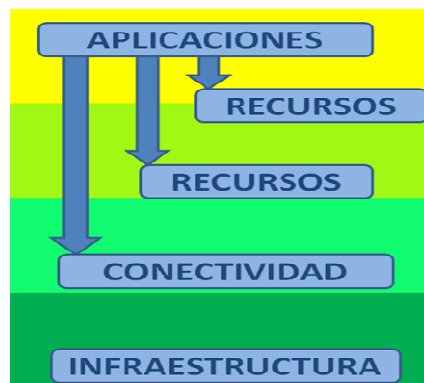
### 2.6.4. ARQUITECTURA DE LA COMPUTACIÓN EN MALLA

Principalmente, la arquitectura propuesta es una arquitectura de protocolos que definen los mecanismos básicos que permiten a los usuarios y a los recursos negociar, establecer, gestionar y explotar la compartición de recursos. Una arquitectura abierta basada en un estándar facilita la



extensibilidad, la interoperabilidad, la portabilidad y la compartición de código. De esta manera la estandarización de los protocolos permitirá regular los servicios y mejorar las capacidades de la Grid. En el nivel de infraestructura se encuentran los recursos computacionales, como son los ordenadores, los clusters, las supercomputadoras, los sistemas de almacenamiento en red, las bases de datos, etc. También se incluyen en este nivel la infraestructura de la red y sus mecanismos de gestión y control. En la terminología Grid, la infraestructura se denomina la fábrica y suministra los componentes que serán compartidos. (Véase Figura 2.7).

El nivel de conectividad incluye los protocolos de comunicación y seguridad que permiten a los recursos computacionales comunicarse. Entre estos protocolos se encuentran: la pila de protocolos TCP/IP, el protocolo SSL, Certificados X.509.



**Figura 2.7.** Arquitectura de la Grid Computing

El nivel de recurso se centra en la gestión de un único recurso y permite tener información y control sobre el mismo. En este nivel se encuentran los protocolos que permiten obtener la información de un recurso: las características técnicas, la carga actual, el precio, etc. También se encuentran los protocolos que permiten el control del recurso: el acceso al mismo, el arranque de procesos, la gestión, la parada, la monitorización, la contabilidad de uso y la auditoría del recurso [15].

La capa de recursos engloba todos los servicios que permiten gestionar un conjunto de recursos. Se encuentran los servicios de directorio, que permiten localizar los recursos que son de nuestro interés; los planificadores distribuidos, que permiten asignar las tareas a cada recurso; la monitorización y diagnóstico de la ejecución de las distintas tareas en que se distribuyen la





ejecución de una aplicación; la contabilidad, que permite calcular el coste de la utilización de varios recursos heterogéneos, y el acceso a datos distribuidos, que gestiona la replicación de datos.

El servicio planificador distribuido es una de las aplicaciones más complejas de un desarrollo Grid, ya que existen planificadores distintos: El planificador que maximiza el uso de los recursos, el planificador de la aplicación que divide la aplicación en tareas, el planificador que asigna los recursos para su ejecución y vigila el desarrollo de los mismos. Los dos primeros verifican la eficiencia del sistema Grid, mientras que el tercero verifica la eficiencia de la aplicación.

El último nivel, el de aplicación, se centra en la definición de protocolos, que permiten a las aplicaciones el acceso a la infraestructura del Grid a través de las distintas capas. Según el tipo de aplicación puede ser necesario conectarse a las distintas capas o acceder directamente a una de ellas, incluso directamente a la infraestructura de trabajos (job scheduler) que intenta maximizar la cantidad de trabajo realizado (trabajos por unidad de tiempo).

### **2.6.5. LO QUE LA COMPUTACIÓN EN MALLA PUEDE HACER**

#### **2.6.5.1. APROVECHANDO LOS RECURSOS QUE NO SIEMPRE SE USAN**

El uso más fácil del Grid Computing es ejecutar una aplicación existente en una máquina diferente [16] [14] [12] [17].

La máquina en que la aplicación normalmente se ejecuta podría estar inusualmente ocupada debido a un pico inusual de actividad. El trabajo en cuestión podría ejecutarse en otra parte en una máquina ociosa en el Grid.

Hay dos requisitos previos a considerar. Primero, la aplicación debe ser ejecutable remotamente. Segundo, la máquina remota debe encontrar cualquier hardware especial, software, o requerimientos de recursos impuestos por la aplicación. Por ejemplo, un trabajo en lotes (batch) que consume una cantidad significativa de tiempo, procesando un conjunto de datos de entrada para producir un conjunto de resultados, es quizás el uso más ideal y simple para una Grid. Si las cantidades de entradas y salidas son grandes, más análisis y planeación podrían requerirse para usar eficazmente la Grid para tal trabajo.



### 2.6.5.2. CAPACIDAD DE CPU PARALELA

El potencial para la capacidad de CPU paralela masiva es uno de los rasgos más atractivos de una Grid. Además de necesidades científicas puras, tal poder de cómputo está conduciendo a una nueva evolución en las industrias como el campo bio-médico, planeación financiera, exploración petrolera, etc. El atributo común entre tales usos es que las aplicaciones se han escrito para usar algoritmos que pueden dividirse independientemente en partes de ejecución.

Una aplicación de Grid intensiva de CPU puede pensarse como muchos sub-trabajos más pequeños, cada uno ejecutándose en una máquina diferente en la Grid.

### 2.6.5.3. EL ACCESO A LOS RECURSOS ADICIONALES

Los recursos adicionales pueden proporcionarse en número y capacidad variable. Por ejemplo, si un usuario necesita aumentar su ancho de banda total a Internet para implementar un mecanismo de búsqueda de minería de datos, el trabajo podría dividirse entre máquinas de la Grid que tienen conexiones independientes a Internet.

### 2.6.5.4. BALANCEO DE RECURSOS

Para aplicaciones habilitadas, la Grid puede ofrecer un efectivo balanceo de recursos mediante la planificación de trabajos de Grid en máquinas con poca utilización, (Véase Figura 2.8).

Esta facilidad puede mejorar invaluablemente el manejo de picos de carga de actividad en sectores de una organización más grande. Esto puede pasar de dos maneras:

- Un pico inesperado puede ser conducido a máquinas relativamente ociosas en la Grid.
- Si la Grid ya se utiliza totalmente, el trabajo de prioridad más baja que se realiza en la Grid debe ser suspendido temporalmente o incluso cancelado y realizado posteriormente para dejar lugar a un trabajo de prioridad mayor.

Sin una infraestructura de Grid, tales decisiones de equilibrio serían difíciles de priorizar y ejecutar.

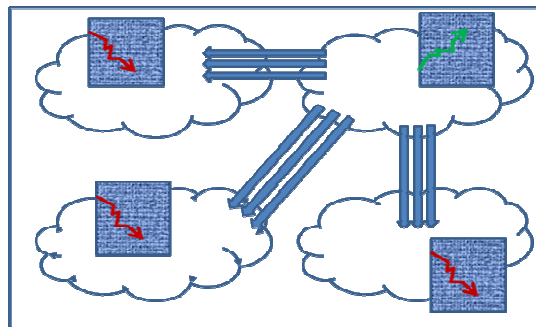


Figura 2.8. Balanceo de recursos.

## 2.7..NET FRAMEWORK REMOTING

### 2.7.1. INTRODUCCIÓN

El framework .NET Remoting, (Véase Figura 2.9), permite crear fácilmente aplicaciones ampliamente distribuidas, tanto si los componentes de las aplicaciones están todos en un equipo, como si están repartidos por el mundo. Se pueden crear aplicaciones de cliente que utilicen objetos en otros procesos del mismo equipo o en cualquier otro equipo disponible en la red. También se puede utilizar .NET Remoting para comunicarse con otros dominios de aplicación en el mismo proceso. También .NET Remoting proporciona un enfoque abstracto en la comunicación entre procesos que separa el objeto utilizado de forma remota de un dominio de aplicación de cliente o servidor específico, y de un mecanismo específico de comunicación. Por lo tanto, se trata de un sistema flexible y fácilmente personalizable. Se puede reemplazar un protocolo de comunicación con otro o un formato de serialización con otro, sin tener que recompilar el cliente ni el servidor. Además, el sistema de interacción remota no presupone ningún modelo de aplicación en particular. Se puede comunicar desde una aplicación Web, una aplicación de consola, un servicio de Windows o desde casi cualquier aplicación que se desee utilizar. Los servidores de interacción remota también pueden ser cualquier tipo de dominio de aplicación. Cualquier aplicación puede albergar objetos de interacción remota y proporcionar sus servicios a cualquier cliente en su equipo o red [34].

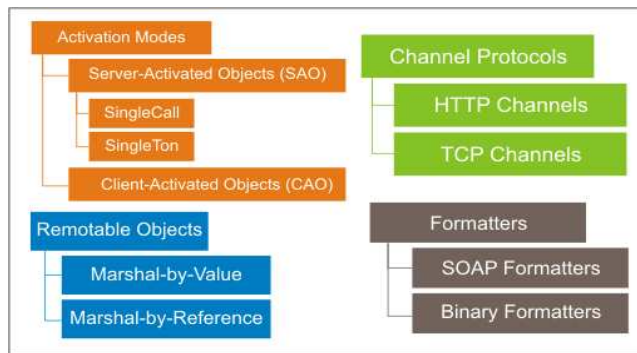


Figura 2.9. Elementos que componen el Framework de .NET Remoting

### 2.7.2. ARQUITECTURA DE .NET FRAMEWORK REMOTING

La verdadera ventaja del sistema de interacción remota es su capacidad para permitir la comunicación entre objetos pertenecientes a dominios de aplicación o a procesos distintos mediante diferentes protocolos de transporte, formatos de serialización, esquemas de duración de objetos y modos de creación de objetos (Véase Figura 2.10). Además, la interacción remota permite intervenir en prácticamente todas las fases del proceso de comunicación, sea cual sea la razón [35].

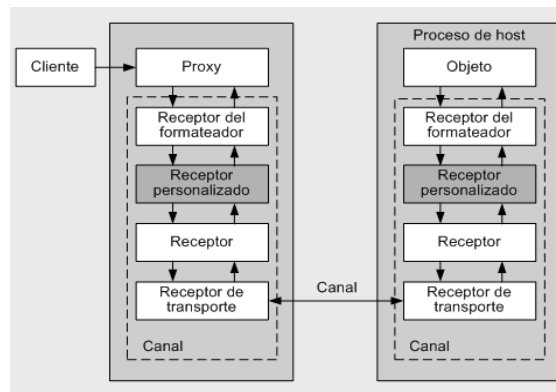


Figura 2.10. Arquitectura del Framework .NET Remoting

#### 2.7.2.1. COPIAS Y REFERENCIAS

La comunicación entre procesos requiere un objeto servidor cuya funcionalidad esté a disposición de los llamadores fuera de su proceso, un cliente que realice llamadas al objeto servidor y un mecanismo de transporte que lleve las llamadas de un extremo a otro. Las direcciones de los métodos del servidor son lógicas y funcionan correctamente en un proceso, pero no funcionan en



otro proceso de cliente. Para solucionar este problema, el cliente puede llamar a un objeto servidor realizando una copia de todo el objeto y pasándola al proceso de cliente, donde se pueden invocar directamente los métodos de la copia.

### 2.7.2.2. ARQUITECTURA SIMPLIFICADA DE INTERACCIÓN REMOTA

La arquitectura de interacción remota proporciona al programador un procedimiento aún más sencillo. Si configura correctamente el cliente, sólo tiene que crear una nueva instancia del objeto remoto mediante **new** (o la función de creación de instancias del lenguaje de programación administrado que utilice). Su cliente recibe una referencia al objeto de servidor, lo que le permite llamar a sus métodos como si el objeto estuviera en su proceso en lugar de estar ejecutándose en otro equipo. El sistema de interacción remota utiliza objetos proxy para dar la impresión de que el objeto del servidor se encuentra en el proceso del cliente. Los objetos proxy son objetos complementarios, que se presentan como si fueran otro objeto. Cuando un cliente crea una instancia del tipo remoto, la infraestructura de interacción remota crea un objeto proxy que, para su cliente, tiene exactamente la misma apariencia que el tipo remoto. Su cliente llama a un método en ese objeto proxy y el sistema de interacción remota recibe la llamada, la dirige hacia el proceso del servidor, invoca al objeto de servidor y envía el valor devuelto al objeto proxy del cliente, que a su vez devuelve el resultado al cliente. En el sistema .NET Remoting, la combinación de tecnologías subyacentes necesarias para abrir una conexión de red y utilizar un determinado protocolo para enviar los bytes a la aplicación receptora se representa como un canal de transporte.

### 2.7.2.3. DISEÑO COMPLETO DE UN SISTEMA DE INTERACCIÓN REMOTA

Para poder citar un ejemplo tenemos que imaginar que deseamos utilizar la funcionalidad de .NET Remoting para realizar la comunicación hacia otro equipo. (Véase Figura 2.11).

Un cliente se limita a crear una nueva instancia de la clase de servidor. El sistema de interacción remota crea un objeto proxy que representa a la clase y devuelve al objeto del cliente una referencia al objeto proxy. Cuando un cliente llama a un método, la infraestructura de interacción remota controla la llamada, comprueba el tipo de información y dirige la llamada por el canal hacia el proceso del servidor. Un canal a la escucha detecta la solicitud y la reenvía al sistema de interacción remota del servidor, que a su vez busca (o crea, si es necesario) y llama al objeto solicitado. A continuación el proceso se invierte: el sistema de interacción remota del servidor incluye la respuesta en un mensaje que el canal del servidor envía al canal del cliente. Por último, el sistema



de interacción remota del cliente devuelve el resultado de la llamada al objeto del cliente a través del objeto proxy.

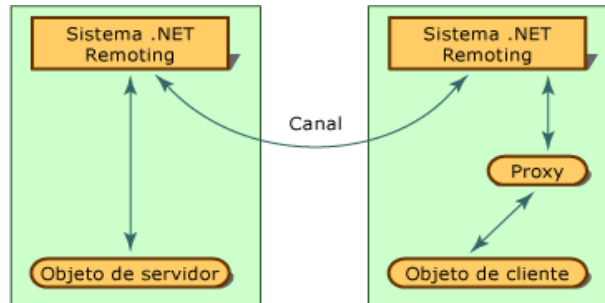


Figura 2.11. Proceso de interacción remota

### 2.7.3. CONVERTIR OBJETOS EN OBJETOS UTILIZABLES DE FORMA REMOTA

#### 2.7.3.1. PROCESOS Y DOMINIOS DE APLICACIÓN

Los sistemas operativos y los entornos de motores de tiempo de ejecución modernos necesitan proteger cada aplicación frente a los errores de las demás aplicaciones. Este mecanismo de protección se implementa mediante el uso de procesos y dominios de aplicación [36].

#### 2.7.3.2. PROCESOS

Para proteger a unas aplicaciones de otras, en los sistemas operativos Microsoft Windows se ejecutan cada una en su propio proceso. Si se produce un error en una aplicación por algún motivo, sólo se ve afectado ese proceso, mientras que las aplicaciones de otros procesos siguen funcionando. Naturalmente, debido a que las direcciones de memoria en un proceso no tienen sentido en ningún otro, puede resultar un tanto complejo llamar a las funciones de un proceso desde otro. Cálculo de referencias es el término asignado a los eventos que se producen cuando una llamada y sus argumentos se empaquetan en un proceso y se desempaquetan en otro, de manera que una llamada que atravesase el límite de un proceso pueda realizarse.



### 2.7.3.3. DOMINIOS DE APLICACIÓN

En el entorno administrado, los dominios de aplicación, que se pueden considerar como procesos lógicos, y los contextos proporcionan aislamiento y seguridad a un costo menor y con una capacidad mayor para escalar correctamente que un proceso de sistema operativo, gracias, entre otros factores, al hecho de que el código administrado dispone de seguridad de tipos verificable. Toda aplicación administrada se ejecuta en un dominio de aplicación, tanto si otra aplicación inicia un dominio en su lugar como si el entorno host inicia uno por ella. .NET Remoting facilita la infraestructura para comunicarse entre dominios de aplicación de una manera sencilla, protegida por las tecnologías de seguridad.

### 2.7.4. PERIODOS Y ACTIVACIÓN DE OBJETOS REMOTOS

En el momento de crear un objeto remoto, tiene que saber cuándo y cómo se crea y se inicializa un nuevo objeto: cómo se activa. Dado que el sistema de interacción remota siempre necesita saber qué tipo de activación es necesario para poder poner los objetos a disposición de los clientes [38].

#### 2.7.4.1. ACTIVACIÓN DE OBJETOS REMOTOS

Existen dos tipos de activación para los objetos de cálculo por referencia:

- Activación por el servidor.

Los objetos activados por el servidor los crea el servidor únicamente cuando son necesarios. No se crean al crear el proxy de cliente llamando a **new**, sino cuando el cliente invoca al primer método de dicho proxy.

Se utiliza la enumeración `WellKnownObjectMode` para configurar los objetos activados en el servidor como objetos **Singleton** o **SingleCall**. Los objetos **Singleton** son aquellos para los que siempre habrá una sola instancia, independientemente de cuántos clientes haya para ese objeto y de cuáles tengan una duración predeterminada (el cliente puede usar el sistema de concesión del período para participar en la duración de las instancias **Singleton**). Cuando se configura un objeto como **SingleCall**, el sistema crea un objeto nuevo por cada llamada a un método de un cliente. Dado que un cliente recibe una referencia a una nueva instancia con cada llamada, los tipos **SingleCall** no participan en el sistema de concesión del período de duración.



- Activación por el cliente.

Los objetos activados en el cliente se crean en el servidor cuando el cliente llama a **new**. El propio cliente, usando el sistema de concesión del período de duración, puede participar en la duración de estas instancias.

### 2.7.4.2. PERIODOS

Los objetos MBR (Marshal-By-Reference, cálculo por referencia) no residen en la memoria eternamente, tanto si son objetos **Singleton** activados en el servidor como si son objetos activados en el cliente. En cambio, a no ser que el tipo reemplace a `MarshalByRefObject.InitializeLifetimeService` para controlar sus propias directivas referentes a la duración, cada objeto MBR tiene una duración controlada por una combinación de concesiones, un administrador de concesiones y una serie de patrocinadores. (En este caso, la duración de un objeto MBR equivale al tiempo total que el objeto permanece activo en la memoria.) Una concesión es el período que un determinado objeto está activo en la memoria antes de que el sistema .NET Framework Remoting comience el proceso para eliminarlo y recuperar la memoria. El administrador de concesiones del dominio de aplicación de servidor es el objeto que determina cuándo el objeto remoto debe ser marcado por el recolector de elementos no utilizados. Un patrocinador es un objeto que solicita una nueva concesión para un determinado objeto, para lo que se registra él mismo en el administrador de concesiones.

Puesto que la vida útil de un objeto remoto es independiente a la de sus clientes, la concesión para un objeto sencillo o pequeño puede ser muy larga, la pueden utilizar varios clientes y la puede renovar periódicamente un administrador o un cliente. Este enfoque utiliza las concesiones de manera eficaz, porque se necesita muy poco tráfico en la red para la recolección distribuida de elementos no utilizados. Sin embargo, los objetos remotos que utilizan recursos escasos pueden tener una concesión para un período de duración breve, que el cliente renueva frecuentemente a intervalos cortos. Cuando todos los clientes han terminado con el objeto remoto, el sistema .NET Framework Remoting elimina rápidamente el objeto. Con esta táctica, en lugar de aumentar el tráfico en la red, se utilizan de una manera más eficaz los recursos del servidor.





### 2.7.5. CANALES

Los canales son objetos que transportan mensajes de una aplicación a otra a través de los límites de interacción remota, tanto de un dominio de aplicación a otro, como de un proceso a otro o de un equipo a otro. Un canal puede escuchar los mensajes entrantes en un extremo, enviar los mensajes salientes a otro extremo o ambas acciones [39].

En el cliente, los mensajes se pasan a la cadena de receptores de canal del cliente después de que hayan recorrido la cadena de contextos del cliente. El primer receptor de canal suele ser un receptor de formato que serializa el mensaje en una secuencia que a continuación, pasa por la cadena de receptores de canal hasta el receptor de transporte del cliente. El receptor de transporte del cliente escribe entonces la secuencia en la conexión.

Los canales deben implementar la interfaz `IChannel`, que proporciona propiedades informativas como `ChannelName` y `ChannelPriority`. Los canales diseñados para estar atentos a la llegada de un determinado protocolo a un determinado puerto implementan `IChannelReceiver` mientras que los canales diseñados para enviar información implementan `IChannelSender`. Tanto el objeto **`TcpChannel`** como el objeto **`HttpChannel`** implementan estas dos interfaces, por lo que se pueden utilizar para enviar o recibir información.

En el servidor, el receptor de transporte del servidor lee las solicitudes de la conexión y pasa la secuencia de solicitud a la cadena de receptores de canal del servidor. El receptor de formato del servidor situado al final de esta cadena deserializa la solicitud en un mensaje. A continuación, lo pasa a la infraestructura de interacción remota.



# Capítulo 3

## *Descripción de la Arquitectura del Sistema*

---

**S**e detalla la arquitectura del sistema donde se describe cada uno de los elementos que la conforman, resaltando a aquellos que realizan el funcionamiento Grid Computing y a los que desarrollan el modelo de paso de mensajes en forma peer to peer para solventar la dependencia de datos, así también la interacción que realizan entre sí.

El contenido de los temas presentados en este capítulo incluye:

- Descripción del marco de trabajo.
- Descripción de la arquitectura del sistema.



### 3.1. DESCRIPCIÓN DEL MARCO DE TRABAJO

Cada día va en aumento la creación de laboratorios virtuales que están siendo montados sobre Internet con la premisa de ofrecer un conjunto de herramientas de Automatización y Diseño Electrónico (EDA) (Véase Figura 3.1). Los desarrolladores de estos laboratorios están combinando interfaces y herramientas amigables para que por medio de estas, los usuarios puedan tener a su disposición todos los medios que tendrían en un laboratorio tradicional, pero con la gran ventaja de que puedan ser utilizados desde cualquier parte del mundo en el horario que se adecue a cada uno de los usuarios. Cada día existen más trabajos que pretenden ofrecer laboratorios que se adecuen a las necesidades de cada usuario, brindándoles como un plus, una forma de experimentación más gráfica e ilustrativa.

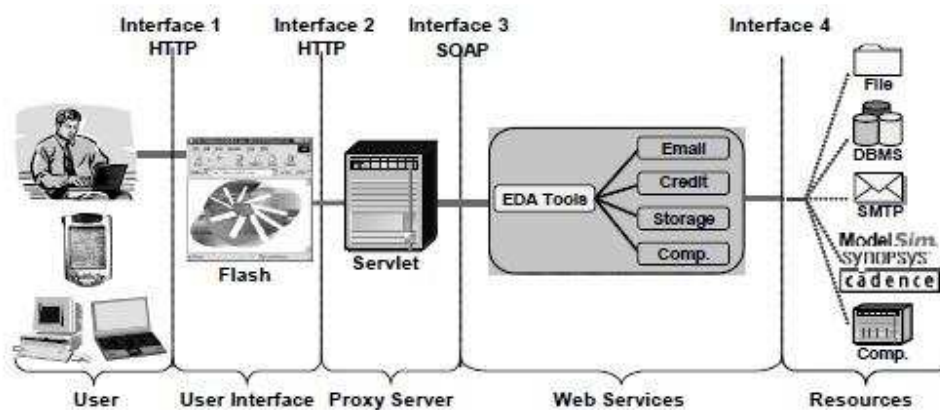


Figura 3.1. Entornos y herramientas que ofrecen los laboratorios virtuales.

Si bien, va en aumento el trabajo que está encaminado al desarrollo de laboratorios virtuales, muy pocos de estos están procurando ofrecer la capacidad de realizar computo distribuido para mejorar el rendimiento en las aplicaciones que requieren una gran cantidad de recursos informáticos y una gran capacidad de poder cómputo. Por este motivo se pretende en este proyecto de investigación enfocarnos a brindar un sistema que les permita a los laboratorios virtuales ejecutar de forma distribuida utilizando al Grid Computing todas aquellas tareas que son demasiado intensivas como para ser resueltas por una sola maquina aislada (Véase Figura 3.2).

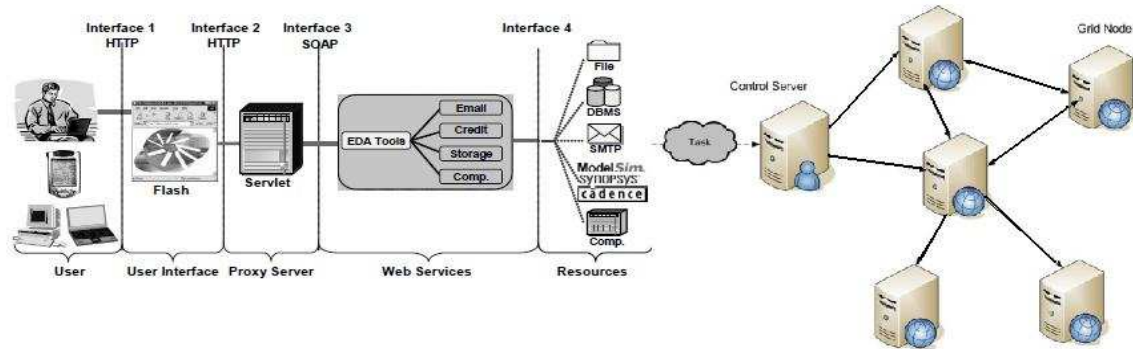


Figura 3.2. Integración del sistema Grid Computing a los laboratorios virtuales.

### 3.2.ARQUITECTURA DEL SISTEMA

En la Figura 3.3, se muestra la arquitectura que conforma al sistema, esta consta de tres elementos principales: 1) Un sistema que crea la plataforma estática del sistema, así como de la presentación de resultados al usuario, estas tareas se realizan cada vez que una nueva aplicación va a ser ejecutada en el sistema, 2) Un maestro que se encarga de realizar el balanceo de cargas de trabajo y la distribución de procesos que conforman a la aplicación que se pretende ejecutar en forma distribuida, 3) De múltiples unidades de ejecución, las cuales a partir de este momento serán llamadas trabajadores, si bien, estos se encargan de realizar la ejecución de los procesos y de solventar la dependencia de datos que exista entre ellos.

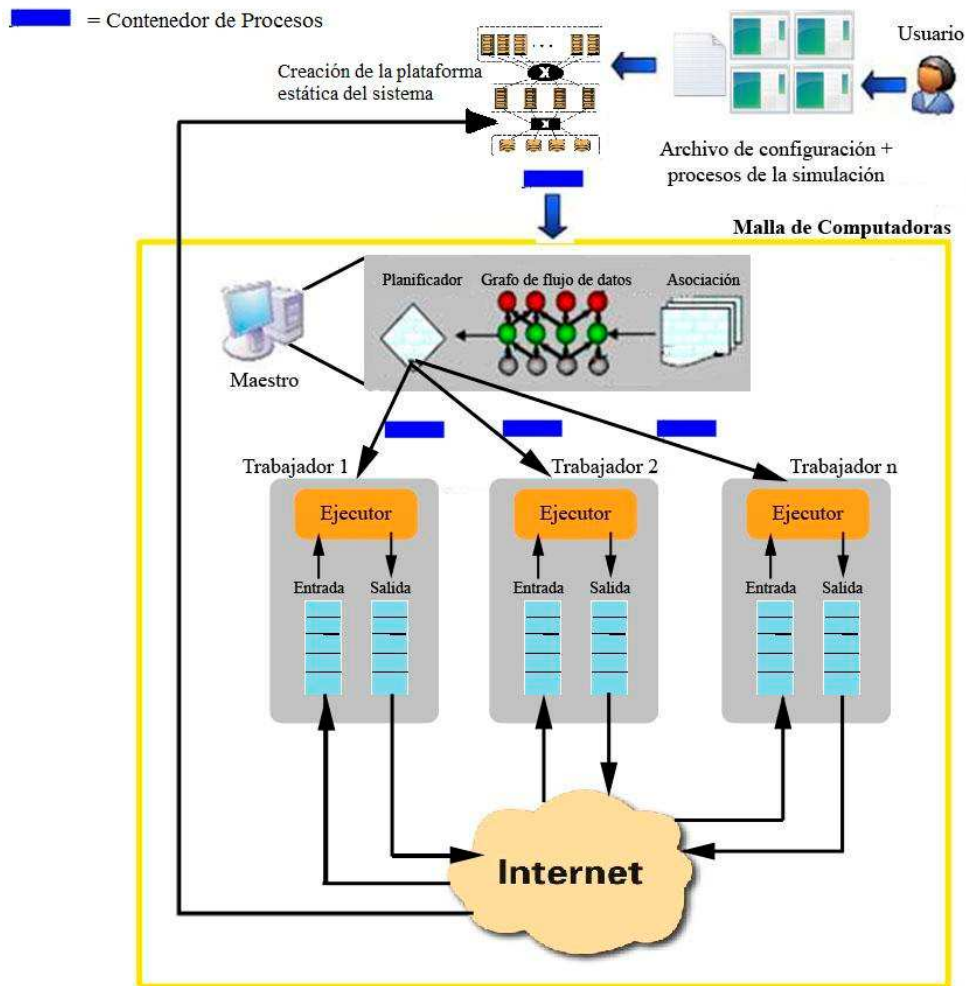


Figura 3.3. Arquitectura del sistema.

### 3.3.SISTEMA PARA LA CREACIÓN DE LA PLATAFORMA ESTÁTICA

Esta parte de la arquitectura comprende de tres elementos los cuales son: El archivo de configuración de la aplicación que va a ser ejecutada en el sistema más los procesos que la conforma. El sistema para la creación de la plataforma estática y por último un contenedor de procesos. Estos elementos se describen a continuación.



### 3.3.1. ARCHIVO DE CONFIGURACIÓN

La estructura del archivo, describe en cada una de sus líneas al grafo de dependencias de los datos, lo que quiere decir, que procesos se comunican entre sí (Véase Figura 3.4). La estructura está definida de la siguiente manera:

En la fila número 1 se indica el PID (Process Identification, Identificador Único de Proceso) con el cual se identificarán a todos los procesos que a una aplicación en concreto que se vaya a ejecutar en el sistema.

A partir de la fila número 2, se indica información perteneciente a cada uno de los procesos; cada campo está separado por comas; donde primeramente se indica el nombre del proceso, posteriormente muestra su peso (requerimientos del proceso), después indica las dependencias entre de datos que tiene con otros procesos. Esto es, el nombre o los nombres de los procesos que necesitarán los resultados que este produzca, a continuación el número total de comunicaciones que habrá entre esos procesos, seguido de esto, muestra la posición de en cola de almacenamiento del proceso, de donde el sistema tendrá que obtener los resultados que necesita otro u otros procesos, por último revela la posición en la cola del proceso en la que tiene que depositar los resultados previamente adquiridos. Repitiendo esta secuencia en la información que se indica para los demás procesos dependientes.

Las siguientes filas en el archivo de configuración contienen información con la misma estructura a la que de explicó en la fila número 2, sólo cambian los nombres de los procesos y sus dependencias. De igual manera, cada archivo de configuración tendrá que tener la misma estructura, ya que el funcionamiento del sistema depende de ello.

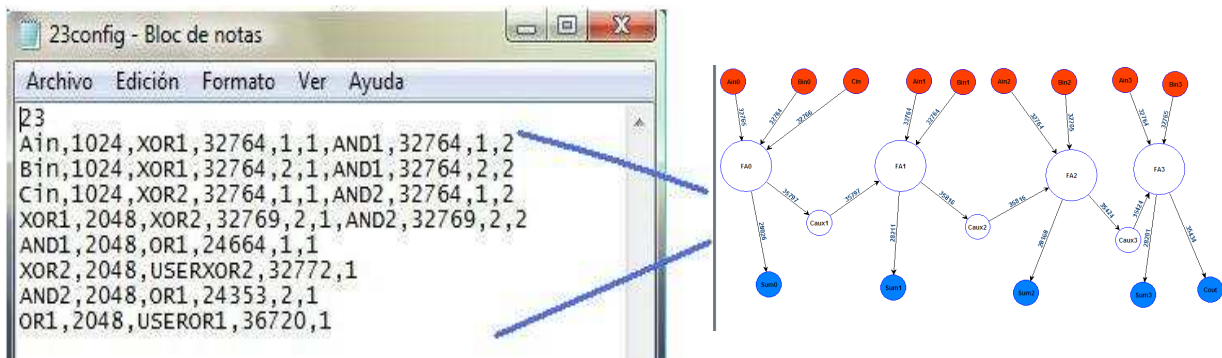


Figura 3.4. Estructura del archivo de configuración.



### 3.3.2. CREACIÓN DE LA PLATAFORMA ESTÁTICA DEL SISTEMA

Es aquí donde se definen y crean los elementos que configuran un ambiente estático para cada ejecución de una aplicación en forma distribuida, a la hora de lanzar la ejecución de una aplicación, la plataforma se adecua para atender las necesidades de la aplicación. La plataforma consta de la creación de una cola de resultados donde se almacenan los resultados que generen ciertos procesos, para que puedan ser observados por el usuario. Además de esto, es aquí donde se obtiene el PID de la aplicación, con el cual se obtienen los eventos que se generen en la bitácora durante la ejecución de esta, logrando con ello la presentación al usuario de todos los eventos generados para el PID de la aplicación que se ejecuta. Por último es en este punto donde se crea un contenedor de procesos el cual contiene toda la información obtenida del archivo de configuración más los procesos que van a ser ejecutados, el cual es enviado al maestro para que se encargue de distribuir el trabajo entre sus trabajadores..

### 3.3.3. CONTENEDOR DE PROCESOS

Es un elemento creado por el sistema para encapsular información y datos los cuales pertenecen a una aplicación que va a ser ejecutada por el sistema. Este contenedor es llenado con la información que se obtiene del archivo de configuración y con los procesos, a continuación se muestra la estructura de este contenedor (Véase Figura 3.5).



Figura 3.5. Contenedor de procesos.

El contenedor de procesos es estructuralmente un tipo de dato perteneciente al lenguaje de programación C# que representa una colección (ArrayList), contiene  $n$  elementos, cada uno de ellos es utilizado dentro la planificación, distribución de procesos y el balanceo de las cargas de trabajo. A continuación se describe cada elemento del contenedor.

- **PID:** El cual nos ayuda a diferenciar las aplicaciones que se estén ejecutando, a pesar de que ciertos procesos pertenecientes a diferentes aplicaciones tengan los mismos nombres.
- **Nombre del proceso:** Este nos ayuda a identificar a los procesos que serán ejecutados.



- **Proceso:** Forma parte de una aplicación, se encarga de crear el sistema de colas, realiza lecturas y escrituras, además de que procesa los datos obtenidos en ellas.
- **Peso del proceso:** Nos indica que procesos generaran más procesamiento que otros, por lo tanto sus requerimientos.
- **Dependencias:** Este campo contiene una serie de información que se encargará de informar al trabajador la dirección IP a donde tiene que transportar los resultados que el proceso en ejecución genere, dicha información está organizada de la siguiente manera:
  - **Campo 1.-**Nombre de la cola de almacenamiento a la que se agregarán los resultados generados por algún proceso en ejecución.
  - **Campo 2.-** Número promedio de conexiones realizadas entre los procesos dependientes.
  - **Campo 3.-** Número de fila de la cual tendrá que escribir el resultado que genero el proceso en cuestión.
  - **Campo 4.-** Número de fila de la cual leerá el resultado generado por el proceso anteriormente mencionado para enviárselo a otro proceso que lo ocupe como entrada.

En este contenedor se almacenan todos los procesos que conforman la aplicación que se va a ejecutar.

### 3.4.MAESTRO

Este es el encargado de obtener recursos de todas aquellas computadoras que formen parte de la red, logrando utilizar la capacidad de procesamiento que tiene cada una de ellas en su beneficio para realizar la ejecución de procesos que pertenecen a aplicaciones intensivas. Es a partir de este elemento donde se puede observar que el sistema procede a interactuar en forma Grid Computing.

El maestro consta de tres componentes los cuales interactúan entre sí para realizar el balanceo de las cargas de trabajo y la distribución de procesos. A continuación se describe cada componente.





### 3.4.1. COMPONENTE ASOCIACIÓN

Se encarga de obtener una lista de los trabajadores disponibles, además de sus recursos, entre los que se encuentran: 1) Espacio libre en la unidad C. 2) Tamaño, velocidad y cantidad de RAM libre. 3) Número de procesadores lógicos y velocidad de cada uno de ellos.

Para lo cual el componente abre el archivo ubicado en la ruta C:/INOUTQUEUES/ConfiguracionTrabajadores/ListaTrabajadores.txt” y obtiene las direcciones IP de todos los trabajadores que se encuentran registrados en ese archivo.

Posteriormente verifica la disponibilidad de cada uno de ellos, para lo cual se encarga de intentar establecer la comunicación con cada uno de los trabajadores. En caso de no poder establecerla con algún Trabajador, el componente procede a descártalo; para todos aquellos trabajadores con los cuales se haya logrado entablar la comunicación, el componente obtiene sus recursos y los agrega a una nueva lista de trabajadores disponibles.

### 3.4.2. COMPONENTE GRAFO DEL FLUJO DE DATOS

Tiene como objetivo crear un grafo donde se encuentran las direcciones IP de todos los trabajadores disponibles, nombres y el número total de procesos asignados a cada trabajador.

El componente clasifica a cada uno de los Trabajadores en: 1) Trabajadores con alto poder de cómputo. 2) Trabajadores con bajo poder de cómputo. Esta clasificación se da por gracias a la obtención de sus recursos mediante el componente asociación la cual se realiza de la siguiente manera: La multiplicación del número de procesadores lógicos por la velocidad de cada uno de ellos más la velocidad de la RAM, esto para cada trabajador, después de realizar dicha operación, verifica que el número obtenido como resultado sea mayor a 900; si es este el caso, se clasifica al trabajador con alto poder de cómputo, en caso contrario con bajo poder de cómputo.

Una realizado lo anterior el componente procede a realizar la clasificación de los procesos, para lo cual verifica el peso de cada uno para obtener la media aritmética de todos los pesos, de esta manera aquel proceso que tenga un peso menor o igual a la media, se clasifica como proceso ligero, en caso contrario se clasifica como proceso pesado.



Por último asigna los procesos a los trabajadores teniendo en cuenta las siguientes situaciones: 1) Siempre asignará un proceso a aquel trabajador que tenga menos procesos asignados para ejecución. 2) Los trabajadores con bajo poder de cómputo sólo se les podrán asignar procesos ligeros, verificando siempre la situación 1. 3) A los trabajadores con alto poder de cómputo se les podrán asignar tanto procesos ligeros como pesados, verificando siempre la situación 1.

### 3.4.3. CONTENEDOR UNITARIO DE PROCESO

El contenedor unitario de proceso al igual que el anterior, es un elemento creado por el sistema para encapsular información y datos pertenecientes a una aplicación que va a ser ejecutada por el sistema; teniendo como diferencia al anterior, que este contenedor es enviado a los trabajadores para que realicen la ejecución del proceso que se encuentra dentro, además sólo consta de cinco campos, donde se incluye información de la ubicación a la que tiene que transportar los resultados que el proceso genere, (Véase Figura 3.6).



Figura 3.6. Estructura del contenedor unitario de proceso.

El contenedor unitario de vértice es estructuralmente un tipo de dato perteneciente al lenguaje de programación C# que representa una colección (ArrayList), contiene 5 elementos, cada uno de ellos es utilizado dentro del proceso de ejecución y almacenamiento de resultados. A continuación se describe cada elemento del contenedor unitario de proceso.

- **PID:** El cual nos ayuda a diferenciar las aplicaciones que se estén ejecutando, a pesar de que ciertos procesos pertenecientes a diferentes aplicaciones tengan los mismos nombres.
- **Nombre del proceso:** Este nos ayuda a identificar a los procesos que serán ejecutados.
- **Proceso:** Forma parte de una aplicación, se encarga de crear el sistema de colas, realiza lecturas y escrituras, además de que procesa los datos obtenidos en ellas.
- **Dependencias:** Este campo contiene una serie de información que se encargará de informar al trabajador la dirección IP a donde tiene que transportar los resultados que el proceso en ejecución genere, dicha información está organizada de la siguiente manera:



- **Campo 1.**-Nombre de la cola de almacenamiento a la que se agregarán los resultados generados por algún proceso en ejecución.
  - **Campo 2.**- Número promedio de conexiones realizadas entre los vértices dependientes.
  - **Campo 3.**- Número de fila de la cual tendrá que escribir el resultado que genero el proceso en cuestión.
  - **Campo 4.**- Número de fila de la cual leerá el resultado generado por el proceso anteriormente mencionado para enviárselo a otro proceso que lo ocupe como entrada.
- **IP del maestro:** La cual se utiliza para se envíen los eventos que se deben almacenar en la bitácora

Cabe mencionar que es la misma estructura que se sigue para todos los contenedores unitarios de proceso.

### 3.4.4. COMPONENTE PLANIFICADOR

Es el encargado de realizar la distribución de procesos y balanceo de las cargas de trabajo entre los trabajadores disponibles pertenecientes a la Grid Computing. Para lo cual crea un contenedor unitario de proceso por cada proceso existente.

Es aquí donde realiza una modificación a un campo 2 de las dependencias, sustituyendo su valor por la dirección IP, ya sea del Trabajador o del Usuario dependiendo de la cola donde se tienen que depositar los resultados; si el nombre de la cola contiene la palabra USER al inicio, se identifica que los resultados se enviarán al Usuario, en caso contrario se enviarán a algún trabajador.

Para finalizar el componente realiza una conexión con cada uno de los trabajadores disponibles para que se realice la entrega del contenedor unitario de proceso.

### 3.5. TRABAJADOR

Es el encargado de realizar la ejecución de los procesos que se asigne el maestro además de ello se encarga de realizar el transporte de los resultados de cada proceso que tenga en ejecución a la cola de resultados de entrada de otros procesos o a la cola de resultados donde se encuentra la interfaz de usuario dependiendo de la dirección IP que se le haya indicado en el campo 2 de las dependencias



que se encuentran dentro del contenedor unitario de proceso. Siendo en esta parte del sistema que se crea el modelo para el paso de mensajes entre procesos utilizando peer to peer.

### 3.5.1. COMPONENTE EJECUTOR

Tiene como objetivo almacenar y lanzar la ejecución de los procesos que recibe para lo cual el componente recibe todos los contenedores unitarios de proceso enviados por el maestro, posteriormente procede a realizar el almacenamiento de los procesos en un directorio de trabajo, así que se posiciona en la ruta "C:/INOUTQUEUES/" donde reconstruye al proceso, una vez terminado lo anterior realiza el envío del evento de almacenamiento del proceso al maestro, para que sea agregado en la bitácora.

En seguida lanza la ejecución del proceso, de igual manera como si se hiciera por medio de la consola DOS. A la hora de lanzar la ejecución del proceso se le pasa como parámetro el PID, ya que cada proceso lo necesita para poder crear los nombres de las colas de entrada y salida para los resultados; después de ello realiza el envío del evento de ejecución del proceso al maestro, para que sea agregado en la bitácora.

También se encarga de transportar los resultados de los procesos que están en ejecución, ya sea que los envíe a otros trabajadores o a la interfaz de usuario dependiendo de lo indicado en el campo de dependencias. Para realizar esta tarea primeramente intenta abrir la cola de resultados de salida generada por el proceso en ejecución, en caso de que no pueda abrirla, ya sea porque aún no está creada o porque se encuentra abierta por otro proceso; el componente espera medio segundo y vuelve a intentar abrirla, realizando este ciclo hasta que lo logre, ya abierta obtiene de cada línea los resultados y los almacena, después borra dichos resultados de la cola, en caso de que alguna línea no contenga ningún resultado, procede nuevamente a esperar medio segundo y después vuelve a realizar las tareas anteriormente mencionadas.

Una vez obtenidos los resultados el componente identifica si el nombre de la cola del proceso dependiente, contiene la palabra USER; en caso afirmativo envía a la interfaz de usuario la siguiente información: 1) Nombre de la cola donde tiene que agregar los resultados. 2) Número de línea donde tiene que escribir los resultados. 3) Los Resultados. 4) El PID. En caso de que no contenga la palabra USER el nombre de la cola del proceso dependiente, se envían los datos mencionados anteriormente al trabajador que está ejecutando el proceso dependiente.

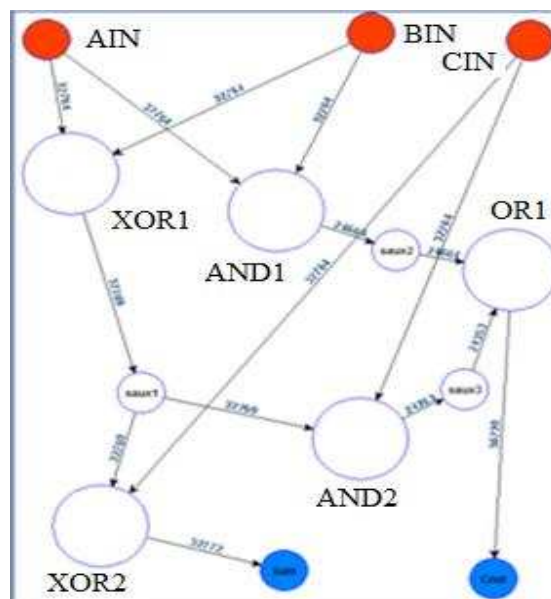


Ya que el trabajador o la interfaz de usuario recibieron los datos junto con la información el componente ejecutor local intenta abrir la cola de resultados del proceso dependiente, en caso de que no pueda abrir la cola, ya sea porque aún no está creada o porque se encuentra abierta por otro proceso; el componente medio segundo y vuelve a intentarlo, realizando este ciclo hasta que logre abrirla, posteriormente ya abierta, agrega los resultados a la cola en la línea indicada.

### 3.5.2. SISTEMA DE COLAS

Este sistema es creado por cada proceso que pertenece a una aplicación que va a ser ejecutada en el sistema, el diseño de estas se da de la siguiente manera:

Cada proceso tiene que crear una cola tanto de entrada como de salida por cada otro proceso que necesite resultados, esto se puede observar de mejor manera por medio de la Figura 3.7, la cual presenta el grafo de comunicaciones que existen entre procesos para una cierta aplicación.



**Figura 3.7.** Grafo que describe la comunicación entre procesos de una aplicación.

Utilizando como un ejemplo para describir de mejor manera la creación de las colas se procede a ocupar la Figura 3.7. El proceso llamado XOR1 tiene que crear una dos colas de entrada para recibir los resultados que le envíen los procesos AIN y BIN y dos colas de salida para los procesos XOR2 y AND2 ya que en estas escribe los resultados que tienen que recibir estos procesos. De igual manera el proceso llamado AND1 debe crear dos colas de entrada para recibir los resultados que



## CAPITULO 3 Descripción de la Arquitectura del Sistema

---



envíen los procesos Ain y BIN y en este caso solo debe crear una cola de salida para el proceso OR1 ya que en estas escribe los resultados que tiene que recibir este proceso. Es importante denotar que se sigue el concepto de creación de colas para cada uno de los demás procesos existentes en grafo; de igual manera para cualquier grafo de comunicaciones que tenga una aplicación.



# Capítulo 4

## *Pruebas y Resultados*

---

**S**e proporciona una perspectiva general de los resultados obtenidos en cuanto al funcionamiento del sistema, además se describen las pruebas realizadas para medir el rendimiento, así como las características de cada equipo de computo utilizado durante el periodo de pruebas.

El contenido de los temas presentados en este capítulo incluye:

- Presentación de resultados.
- Presentación de pruebas que evalúan el rendimiento del sistema.
- Descripción de la generación del benchmark (sumador de 1 bit con acarreo).



### 4.1 PRESENTACIÓN DE RESULTADOS

Con motivo de poder especificar a fondo los resultados que se obtuvieron en cuanto al funcionamiento del sistema, se muestran a través de una prueba realizada. Esta genera la simulación de un circuito digital (sumador de 1 bit con acarreo), este fue particionado en 8 procesos los cuales interactúan entre sí; para esta prueba se ocuparon 3 computadoras participantes con las siguientes características. (Véase Tabla 4.1). Dichas computadoras están conectadas a una LAN.

**Tabla 4.1.** Características de las computadoras pertenecientes a la Grid.

IP	CARACTERISTICAS	DESEMPEÑO DE TRABAJO COMO:
192.168.1.65	-Velocidad de Procesador: <b>2 GHz.</b> -Tamaño de la RAM: <b>2.00 GB.</b> -Velocidad de la RAM: <b>667MHz.</b> -Tamaño del Disco Duro: <b>100 GB.</b> -Sistema Operativo: <b>Windows Vista Home Premium.</b>	<b>Usuario y Maestro</b>
192.168.1.68	-Velocidad de Procesador: <b>400 MHz.</b> -Tamaño de la RAM: <b>200 MB.</b> -Velocidad de la RAM: <b>70MHz.</b> -Tamaño del Disco Duro: <b>40 GB.</b> -Sistema Operativo: <b>Windows 2000 Profesional SP 4.</b>	<b>Trabajador</b>
192.168.1.73	-Velocidad de Procesador: <b>3 GHz</b> -Tamaño de la RAM: <b>256 MB.</b> -Velocidad de la RAM: <b>400 MHz.</b> -Tamaño del Disco Duro: <b>40 GB.</b> -Sistema Operativo: <b>Windows XP SP 2.</b>	<b>Trabajador</b>

#### 4.1.1 CARGA DEL ARCHIVO DE CONFIGURACIÓN DE LA APLICACIÓN Y CREACIÓN DEL CONTENEDOR DE PROCESOS

Como primer paso el sistema se encargada de procesar el archivo de configuración de la simulación, al cual se le asigno el PID 23.





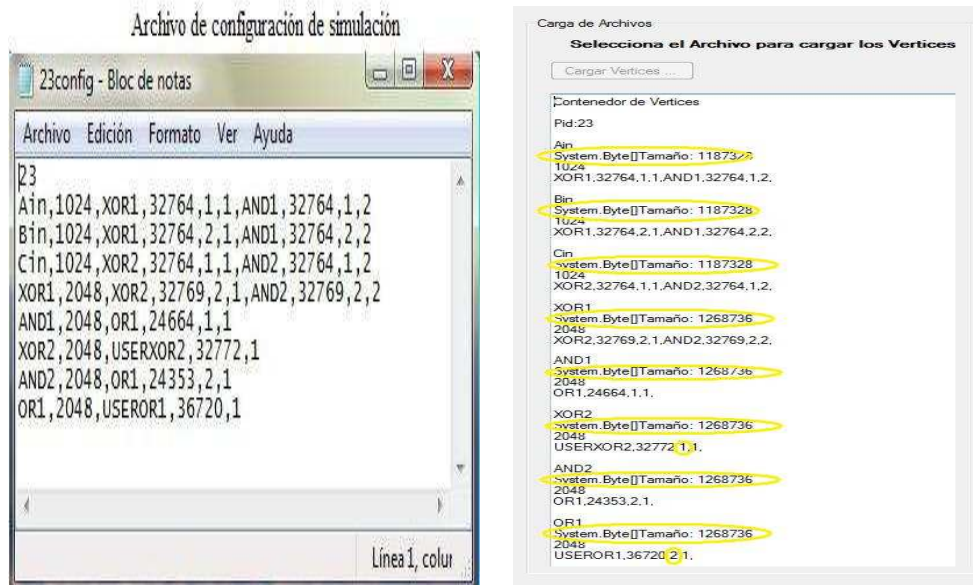
## CAPITULO 4 Pruebas y Resultados



Una vez que se ha obtenido la información del archivo de configuración, el sistema, se encarga de crear un contenedor de procesos donde almacena la información obtenida de dicho archivo más los procesos que conforman a la simulación.

**NOTA:** Cuando el sistema detecta que el nombre de algún proceso tiene al inicio la palabra USER, procede a crear una cola donde se almacenaran los resultados que tal proceso genere, también agrega información en la parte que indica las dependencias de datos que tienen los procesos, al contenedor de procesos para que cuando ejecuten ese proceso, los resultados que genere se almacenen en la cola especificada por el sistema. Esta tarea se realiza en forma dinámica para cada proceso que vaya a generar resultados que el usuario deba observar.

Una vez realizada las tareas anteriormente mencionadas, el sistema procede a mostrar el contenido del contenedor de procesos a través de la interfaz de usuario. En la Figura 4.1 se muestra una comparación del archivo de configuración de la simulación, y el contenido del contenedor de procesos, donde se pueden observar las modificaciones (encerradas en un círculo amarillo), que el sistema realizo a la hora de crear dicho contenedor.



**Figura 4.1.** Comparación entre el contenido del archivo de configuración y el contenido contenedor de vértices.



### 4.1.2 CREACIÓN DE COLA DE RESULTADOS

La creación de la cola de resultados se realiza mientras se está creando el archivo de configuración, ya que es en este punto donde se identifican aquellos procesos que van a entregar resultados para el Usuario (En este caso existen dos que entregan resultados al Usuario: *XOR2* y *OR1*). Tal cola de resultados contiene un nombre compuesto por tres elementos: el *PID* de la simulación más la palabra *Resultados* más el indicativo de que la cola es de entrada (*IN*). Posteriormente a esto, escribe en la cola el nombre del proceso, ya sin la palabra *USER*; esto se realiza para que el usuario sepa qué resultados pertenecen a qué Procesos, la Figura 4.2 muestra la cola de resultados creada por el sistema.



**Figura 4.2.** Cola de resultados perteneciente a la simulación del sumador de 1 bit con acarreo, a la cual se le asigno el PID 23.

### 4.1.3 ENVIÓ DEL CONTENEDOR DE PROCESOS AL MAESTRO

Ya que el contenedor de procesos ha sido creado, se procede a realizar el envío de este al Maestro, por tanto, el sistema verifica si puede establecer comunicación con el Maestro, lo que genera dos situaciones:

- **La comunicación no se logra establecer:** Esto generaría un error el cual se mostraría al Usuario mediante la interfaz de Usuario.
- **La comunicación se establece correctamente:** Lo que permite realizar el envío del contenedor de procesos.



### 4.1.4 PLANIFICACIÓN, DISTRIBUCIÓN DE PROCESOS Y BALANCEO DE LAS CARGAS DE TRABAJO

Estas acciones son realizadas por los componentes pertenecientes al Maestro. Primeramente el sistema se encarga de utilizar el método de planificación (basado en los requerimientos de cada proceso y los recursos disponibles en cada unidad de ejecución); lo que el sistema realiza en este momento es una clasificación de procesos que conforma la simulación basándose precisamente en sus pesos (requerimientos de cada proceso).

El sistema al obtener dichos pesos se encarga de realizar la ordenación en forma ascendente de ellos, para que posteriormente obtenga de todos esos valores la media aritmética, esta se ocupa para realizar la siguiente clasificación:

- Procesos que tengan un peso igual o menor a la media obtenida, se clasifican como procesos LIGEROS.
- Procesos que tengan un peso mayor a la media obtenida, se clasifican como procesos PESADOS.

Esta clasificación es realizada cada vez que una aplicación va a ser ejecutada por el sistema. Para este caso arroja como resultado para la media aritmética el valor de 2048 como lo muestra la Figura 4.3. Es importante denotar que los procesos tienen los siguientes pesos: **Proceso 1:** 1024, **Proceso 2:** 1024, **Proceso 3:** 1024, **Proceso 4:** 2048, **Proceso 5:** 2048, **Proceso 6:** 2048, **Proceso 7:** 2048, **Proceso 8:** 2048.

```
ca. file:///D:/Maestria-CIC/Semestre4/Seminario4/Proyecto/Maestro/ServidorMaestro/bin/Debug/Servi...
Servidor listo para aceptar mensajes...
Pulse INTRO para salir

La media aritmetica obtenida por medio de los pesos de todos los vértices es:
2048
```

**Figura 4.3.** Media aritmética obtenida por los pesos de 8 procesos pertenecientes a la simulación del sumador de 1 bit con acarreo.



### 4.1.5 OBTENCIÓN DE LA LISTA DE TRABAJADORES DISPONIBLES (COMPONENTE ASOCIACIÓN)

Primero el sistema se encarga de obtener por medio de un archivo de texto las direcciones IP de todos los Trabajadores pertenecientes a la Grid, obtiene dichas direcciones, enseguida de ello intenta establecer la comunicación con cada uno de los trabajadores. Para aquellos con los que logre establecer la comunicación, los agrega a una lista junto con sus recursos como lo muestra la Figura 4.4; en el caso de aquellos Trabajadores con los cuales no logró establecer comunicación, el sistema procede a descartarlos.

```
ca: file:///D:/Maestria-CIC/Semestre4/Seminario4/Proyecto/Maestro/ServidorMaestro/bin/Debug/Servi...
Trabajador: 192.168.1.68
13.35126
70
208
84.30859
1
397

Trabajador: 192.168.1.73
20.58281
400
256
95.54688
1
3000
```

**Figura 4.4.** Características de los Trabajadores disponibles pertenecientes a la Grid.

La Figura anterior nos permite observar las características que tiene cada trabajador, mostrándolas en forma de lista: espacio libre en la unidad local C, velocidad de la RAM, Tamaño de la RAM, Tamaño del espacio libre en la RAM, número de procesadores, velocidad del procesador. Estos recursos son utilizados por el sistema para clasificar a los Trabajadores.

### 4.1.6 CREACIÓN DEL GRAFO DE FLUJO DE DATOS (COMPONENTE GRAFO DEL FLUJO DE DATOS)

El sistema se encarga de clasificar a cada trabajador de la siguiente manera:



## CAPITULO 4 Pruebas y Resultados



- **Trabajadores con bajo poder de computo:** Si la multiplicación del número de procesadores por la velocidad de cada uno de ellos, mas la velocidad de la memoria, da como resultado un número  $<$  a 900.
- **Trabajadores con alto poder de computo:** Si la multiplicación del número de procesadores por la velocidad de cada uno de ellos, mas la velocidad de la memoria, da como resultado un número  $>$  a 900.

En la Figura 4.5 se puede observar que el sistema ha clasificado al trabajador 192.168.1.68 con bajo poder de cómputo debido a que la operación aritmética con sus recursos, mencionada anteriormente, no ha superado la cifra de 900.

```
file:///D:/Maestria-CIC/Semestre4/Seminario4/Proyecto/Maestro/ServidorMaestro/bin/Debug/Servi...
trabajadores con ALTO poder de cómputo
192.168.1.73

trabajadores con BAJO poder de cómputo
192.168.1.68
```

**Figura 4.5.** Clasificación de Trabajadores.

Posteriormente asigna los procesos a los trabajadores. Dando como resultado lo que muestra la Figura 4.6, nos permite observar que al Trabajador *192.168.1.68* sólo le han asignado 2 procesos, esto se da debido a que el sistema identificó que los pesos de los procesos *AIN* y *CIN* están por debajo de la media aritmética obtenida anteriormente, por lo cual los clasifíco como ligeros, además de ello, este trabajador fue clasificado con bajo poder de computo, lo que quiere decir que sólo se le van a asignar procesos ligeros. Al trabajador *192.168.1.73* se le asignaron 6 procesos, ya que este fue clasificado con alto poder de cómputo, lo que quiere decir que puede ejecutar tanto procesos ligeros como pesados.



```
file:///D:/Maestria-CIC/Semestre4/Seminario4/Proyecto/Maestro/ServidorMaestro/bin/Debug/Servi...
Contenido del grafo de flujo de datos:
IP Trabajador: 192.168.1.68
Total Vertices en Ejecucion: 2
Nombre de los vertices Asignados a este trabajador: Ain Cin

IP Trabajador: 192.168.1.73
Total Vertices en Ejecucion: 6
Nombre de los vertices Asignados a este trabajador: Bin XOR1 AND1 XOR2 AND2 OR1
```

Figura 4.6. Asignación de procesos a trabajadores.

### 4.1.7 ENVIÓ DE LOS PROCESOS A LOS TRABAJADORES (COMPONENTE PLANIFICADOR)

Es ahora cuando el sistema procede a crear un nuevo contenedor unitario de proceso, por cada proceso a distribuir.

Enseguida, intenta establecer la comunicación con cada trabajador para realizar el envío de los contenedores unitarios de proceso, (Véase Figura 4.7).



```
ca. file:///D:/Maestria-CIC/Semestre4/Seminario4/Proyecto/Maestro/ServidorMaestro/bin/Debug/Servi...
Contenedor unitario de vértice que se envia al Trabajador: 192.168.1.68
PID de la simulacion: 23
Nombre del vértice: Ain
Vertice: System.Byte[]
Dependencias: KOR1,192.168.1.73,1,1,AND1,192.168.1.73,1,2,
Ip del Maestro:192.168.1.65

Contenedor unitario de vértice que se envia al Trabajador: 192.168.1.73
PID de la simulacion: 23
Nombre del vértice: Bin
Vertice: System.Byte[]
Dependencias: KOR1,192.168.1.73,2,1,AND1,192.168.1.73,2,2,
Ip del Maestro:192.168.1.65

Contenedor unitario de vértice que se envia al Trabajador: 192.168.1.68
PID de la simulacion: 23
Nombre del vértice: Cin
Vertice: System.Byte[]
Dependencias: KOR2,192.168.1.73,1,1,AND2,192.168.1.73,1,2,
Ip del Maestro:192.168.1.65

Contenedor unitario de vértice que se envia al Trabajador: 192.168.1.73
PID de la simulacion: 23
Nombre del vértice: KOR1
Vertice: System.Byte[]
Dependencias: KOR2,192.168.1.73,2,1,AND2,192.168.1.73,2,2,
Ip del Maestro:192.168.1.65

Contenedor unitario de vértice que se envia al Trabajador: 192.168.1.73
PID de la simulacion: 23
Nombre del vértice: AND1
Vertice: System.Byte[]
Dependencias: OR1,192.168.1.73,1,1,
Ip del Maestro:192.168.1.65

Contenedor unitario de vértice que se envia al Trabajador: 192.168.1.73
PID de la simulacion: 23
Nombre del vértice: KOR2
Vertice: System.Byte[]
Dependencias: USERKOR2,192.168.1.65,1,1,
Ip del Maestro:192.168.1.65

Contenedor unitario de vértice que se envia al Trabajador: 192.168.1.73
PID de la simulacion: 23
Nombre del vértice: AND2
Vertice: System.Byte[]
Dependencias: OR1,192.168.1.73,2,1,
Ip del Maestro:192.168.1.65

Contenedor unitario de vértice que se envia al Trabajador: 192.168.1.73
PID de la simulacion: 23
Nombre del vértice: OR1
Vertice: System.Byte[]
Dependencias: USEROR1,192.168.1.65,2,1,
Ip del Maestro:192.168.1.65
```

**Figura 4.7.** Creación y envió de los contenedores unitarios de proceso a los trabajadores.

Dentro de cada contenedor se encuentra la información y datos necesarios para que cada trabajador ejecute los procesos que se le asignaron y además de ello envíe los resultados generados por este al trabajador que tenga en ejecución al proceso que necesita estos resultados. Esto se logra gracias a que las dependencias se actualizaron con la dirección IP de donde está el proceso dependiente, de esta manera se realiza la comunicación peer to peer entre trabajadores e interfaz de usuario para la entrega de resultados.

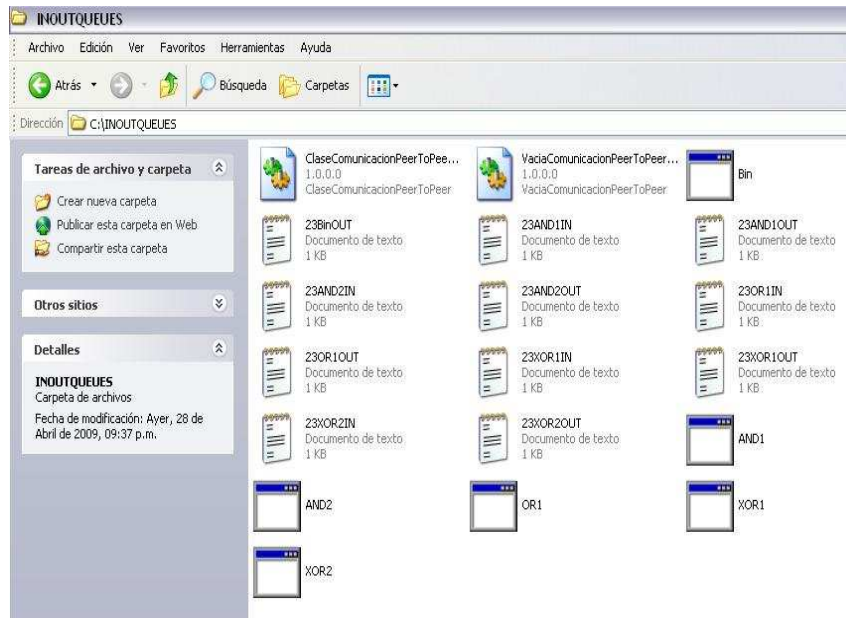




### 4.1.8 EJECUCIÓN DE LOS PROCESOS (COMPONENTE EJECUTOR)

Estas acciones son realizadas por los componentes pertenecientes a los trabajadores. Una vez que un trabajador ha recibido el contenedor unitario de proceso, el sistema extrae los elementos de este contenedor. Enseguida reconstruye el proceso y lo almacena en el directorio “C:/INOUTQUEUES/”, terminada dicha tarea envía el evento de almacenamiento de proceso al Maestro para que este lo almacene en la bitácora.

Dando como resultado que en el Trabajador *192.168.1.73* se almacenaron 6 procesos los cuales crearon sus colas tanto de entrada como de salida. Por tanto en el Trabajador *192.168.1.68* se almacenaron los 2 procesos restantes (Véase Figura 4.8 y 4.9).



**Figura 4.8.** Almacenamiento de los procesos en el trabajador *192.168.1.73*.





## CAPITULO 4 Pruebas y Resultados

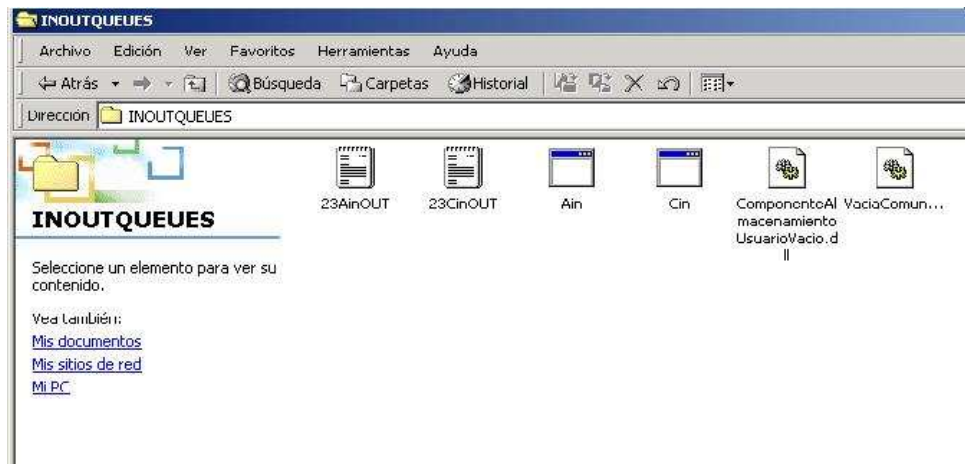


Figura 4.9. Almacenamiento de los procesos en el trabajador *192.168.1.68*.

Una vez realizado lo anterior, el sistema procede a lanzar la ejecución de los procesos, a los cuales les pasa como parámetro el PID de la simulación al que pertenecen. Además de ello envía el evento de ejecución al Maestro para que sea almacenado en la bitácora.

En la Figura 4.10 se puede observar que los procesos comienzan a tratar de leer o escribir en sus respectivas colas, en las cuales se escriben los datos que cada proceso tiene que computar un momento después de que el sistema lanzo su ejecución.

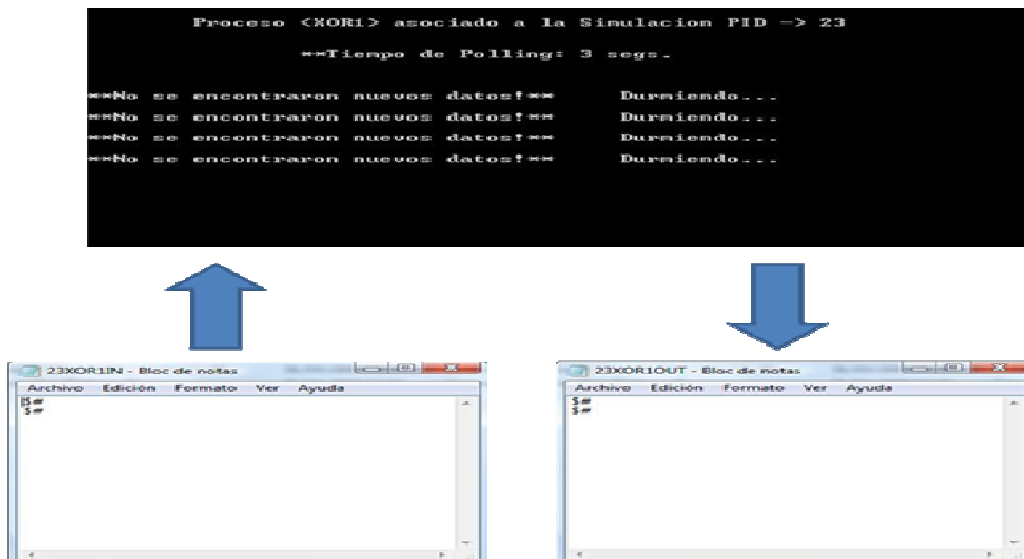


Figura 4.10. Lectura y escritura en las colas de cada proceso.

Una vez hecho lo anterior el sistema envía el evento de ejecución del proceso al Maestro.



### 4.1.9 SOLUCIONANDO LA DEPENDENCIAS DE DATOS.

El sistema obtiene los resultados de la cola de salida de cada proceso y los envía a la dirección IP que se le indico en las dependencias que se encuentran dentro del contenedor unitario de proceso. Para que sean almacenados en la cola de entrada del proceso que utiliza dichos resultados (Véase Figura 4.11). Es en este punto donde el transporte de resultados, ya sea entre trabajadores o trabajadores – interfaz de usuario se realiza en forma peer to peer.

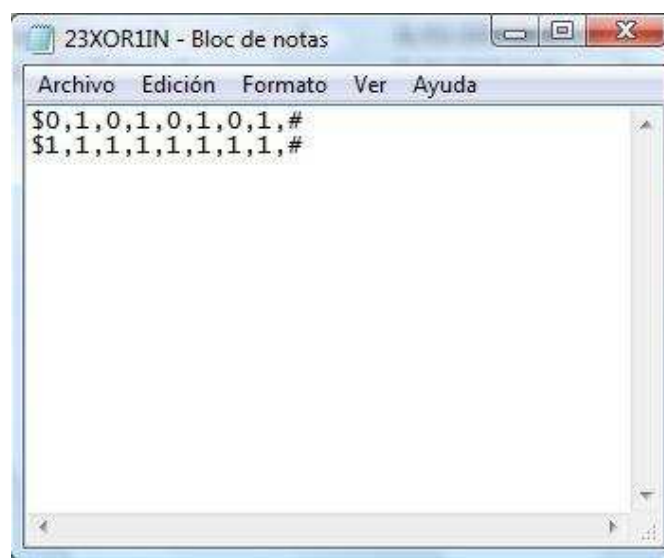
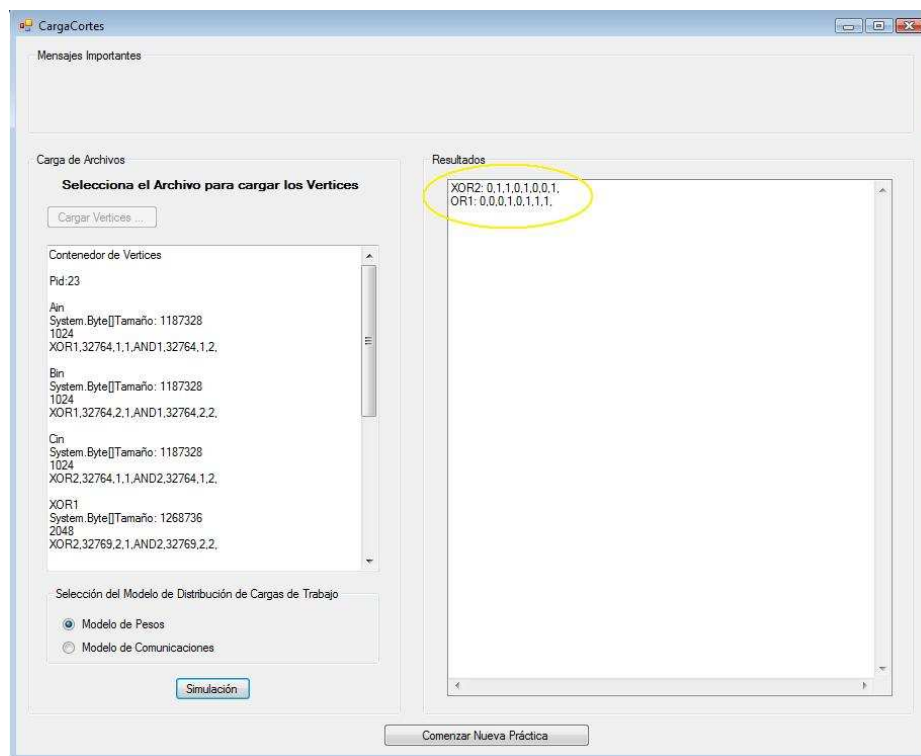


Figura 4.11. Agregado de los resultados en la cola de entrada de un proceso.

### 4.1.10 PRESENTACIÓN DE RESULTADOS AL USUARIO

En este punto el sistema se encarga copiar el contenido de la cola de resultados para posteriormente mostrarlo a través de la interfaz de usuario, logrando con ello que el usuario pueda observar los resultados que se vayan generando (Véase Figura 4.12).



**Figura 4.12.** Presentación de resultados al Usuario.

### 4.1.11 PRESENTACIÓN DE LA BITÁCORA AL USUARIO

Como primer paso sistema obtiene el PID de la simulación ya que gracias a este se pueden recuperar todos los eventos generados durante la simulación. Como resultado, se obtienen todos los eventos registrados en la bitácora para el PID 23, el cual conformó la simulación del sumador de 1 bit con carro. (Véase Figura 4.13).

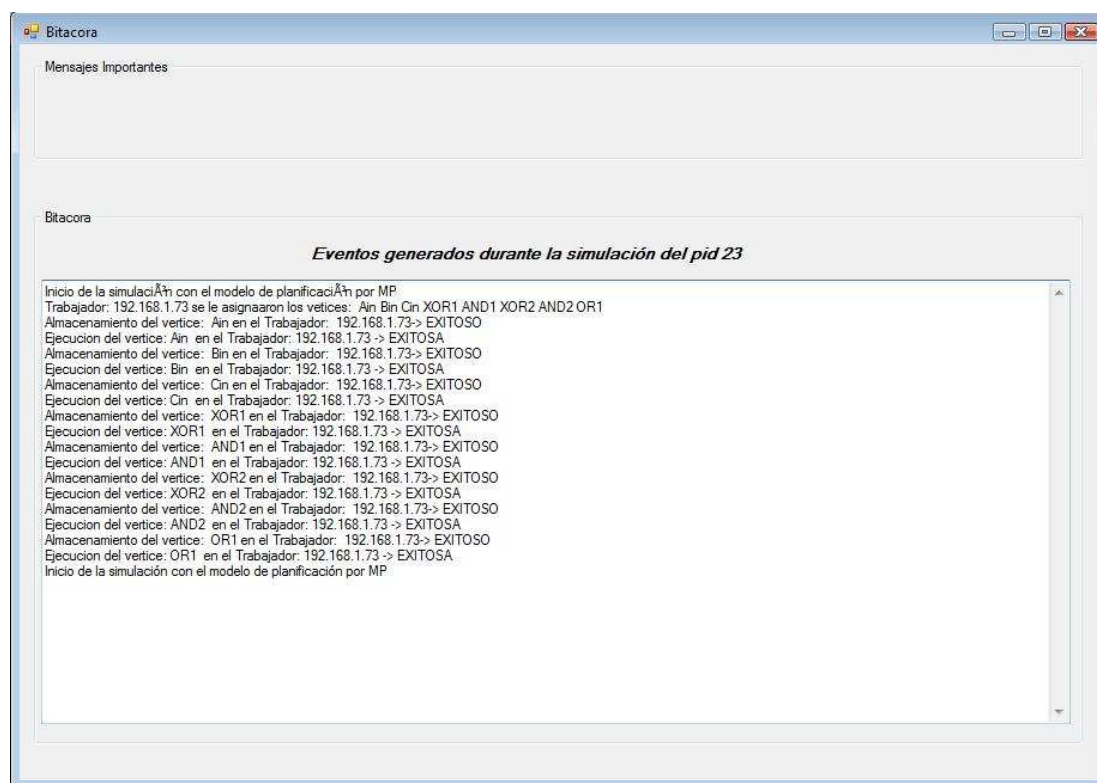


Figura 4.13. Presentación de la bitácora al Usuario.

## 4.2 PRESENTACIÓN DE PRUEBAS QUE EVALÚAN EL RENDIMIENTO DEL SISTEMA

### 4.2.1 CARACTERÍSTICAS DE LOS EQUIPOS DE CÓMPUTO QUE CONFORMAN LA GRID.

En la tabla 4.2 se describen las características de los equipos que conformaron la Grid.

Tabla 4.2. Características de los equipos de cómputo pertenecientes a la Grid.

IP	CARACTERÍSTICAS	DESEMPEÑO DE TRABAJO COMO:
	-Velocidad de Procesador: <b>2 GHz.</b> -Tamaño de la RAM: <b>2.00 GB.</b> -Espacio libre en la unidad C: <b>23.8 GB.</b> -Sistema Operativo: <b>Windows Vista Home Premium.</b>	<b>Usuario y Maestro</b>
148.204.64.209	-Velocidad de Procesador: <b>3 GHz.</b>	<b>Trabajador</b>



## CAPITULO 4 Pruebas y Resultados



	-Tamaño de la RAM: <b>1.00 GB.</b> -Tamaño de la unidad C: <b>54.6 GB.</b> -Sistema Operativo: <b>Windows XP Profesional SP2.</b>	
148.204.64.206	-Velocidad de Procesador: <b>3 GHz.</b> -Tamaño de la RAM: <b>1.00 GB.</b> -Tamaño de la unidad C: <b>46.9 GB.</b> -Sistema Operativo: <b>Windows XP Profesional SP2.</b>	Trabajador
148.204.64.203	-Velocidad de Procesador: <b>3 GHz.</b> -Tamaño de la RAM: <b>1.00 GB.</b> -Tamaño de la unidad C: <b>66.5 GB.</b> -Sistema Operativo: <b>Windows XP Profesional SP2.</b>	Trabajador
148.204.64.205	-Velocidad de Procesador: <b>2.53 GHz.</b> -Tamaño de la RAM: <b>1.00 GB.</b> -Tamaño de la unidad C: <b>97.6 GB.</b> -Sistema Operativo: <b>Windows XP Profesional SP2.</b>	Trabajador
148.204.64.199	-Velocidad de Procesador: <b>2.53 GHz.</b> -Tamaño de la RAM: <b>1.00 GB.</b> -Tamaño de la unidad C: <b>149 GB.</b> -Sistema Operativo: <b>Windows XP Profesional SP2.</b>	Trabajador
148.204.64.197	-Velocidad de Procesador: <b>3 GHz.</b> -Tamaño de la RAM: <b>1.00 GB.</b> -Tamaño de la unidad C: <b>66.5 GB.</b> -Sistema Operativo: <b>Windows XP</b>	Trabajador

### 4.2.2 DESCRIPCIÓN DEL DESARROLLO DE LA APLICACIÓN (SUMADOR DE 1 BIT CON ACARREO) UTILIZADA EN LAS PRUEBAS

#### DESCRIPCIÓN DE LA ARQUITECTURA.

El flujo de esta metodología puede ser descrita en 4 bloques principales:

- Código del diseño a ser verificado en lenguaje SystemC.
- Pre-simulación para monitoreo y construcción de estadística.
- Construcción del grafo que describe al sistema.



- Primer algoritmo de particionamiento por módulo.

En seguida describo brevemente las funciones de cada bloque, considerando sus entradas y sus salidas para poder construir al final un mapa que nos permita figurar el esquema global de trabajo.

### CÓDIGO EN SYSTEMC.

Los lenguajes de descripción de hardware (HDL) nos se han utilizado tanto para diseños arquitecturas en sistemas digitales como referencia en el análisis y validación de estos circuitos. Ofrecen una metodología de diseño jerárquica en las modalidades de *bottom-up* y *top-down*. En particular el lenguaje SystemC, permite programar en diferentes niveles de abstracción: nivel compuerta, nivel transferencia de registros, nivel comportamiento, nivel transaccional, etc. Debido a que el nivel comportamiento representa un nivel aceptable (ni tan abstracto ni tan detallado) de programación (permite estructuras *case*, *if*, *for*), actualmente la mayoría de diseños de circuitos se realiza utilizando este nivel de abstracción.

En este proyecto aprovecharemos las ventajas del diseño jerárquico de los lenguajes de descripción de hardware en particular **SystemC**.

En esencia, SystemC es una biblioteca estandarizada por el IEEE portable en C++ y usada para modelar sistemas con comportamiento concurrente. SystemC suministra mecanismos valiosos para modelar hardware mientras se usa un ambiente compatible con el desarrollo de software. Este lenguaje también provee estructuras orientadas a modelar hardware que no están disponibles en los lenguajes de programación normales.

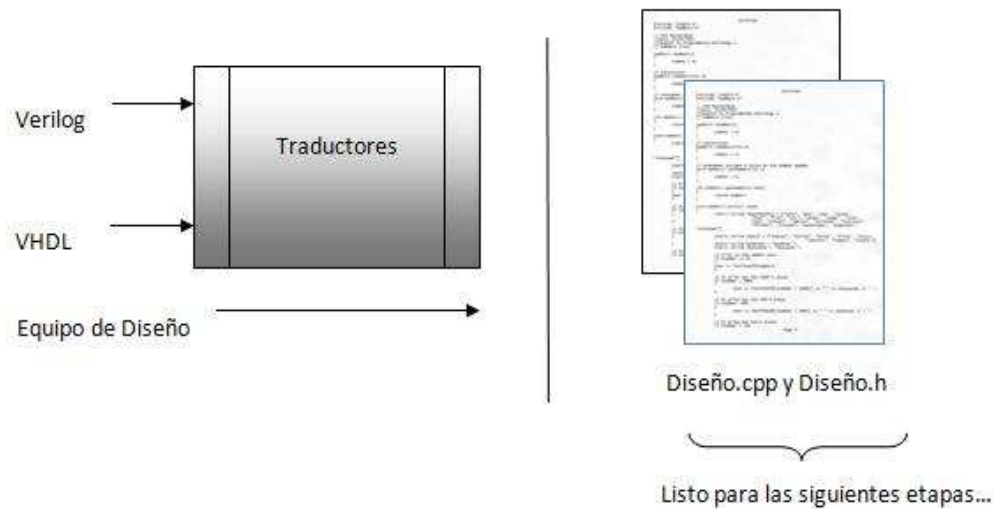
El motor de ejecución de SystemC permite la simulación de procesos concurrentes al igual que Verilog (Lenguaje de descripción de hardware), VHDL (Lenguaje de descripción de hardware) o cualquier otro lenguaje de descripción de hardware. Esto es posible gracias al uso de un modelo cooperativo multitarea, el cual corre en un kernel que orquesta el intercambio de procesos y mantiene los resultados y evaluaciones de dichos procesos alineados en tiempo.

Por otro lado, ya que C++ implementa la orientación a objetos (paradigma que de hecho fue creado para técnicas de diseño de Hardware) la abstracción de datos, propiedades, comportamientos y atributos de un modulo (clase) puede ser el principio para elaborar un diseño jerárquico.



A su vez, es posible describir el comportamiento de las entidades desde varios niveles de abstracción y puede ser explotado para obtener flujos de diseño y verificación más rápidos.

Una vez expuesto lo anterior, hay varias rutas por las cuales podemos llegar a tener el código de un sistema en SystemC listo para ser simulado: Traducido de algún otro lenguaje como VHDL o Verilog, o escrito directamente en SystemC. (Véase Figura 4.14).



**Figura 4.14.** Conversión de código Verilog, VHDL a SystemC.

Hasta ahora hemos probado varias alternativas de traducción y para Verilog, los traductores V2SC de Mazdak & Alborz Design Automation y Verilator de Veripool han pasado satisfactoriamente las traducciones de Verilog a SystemC.

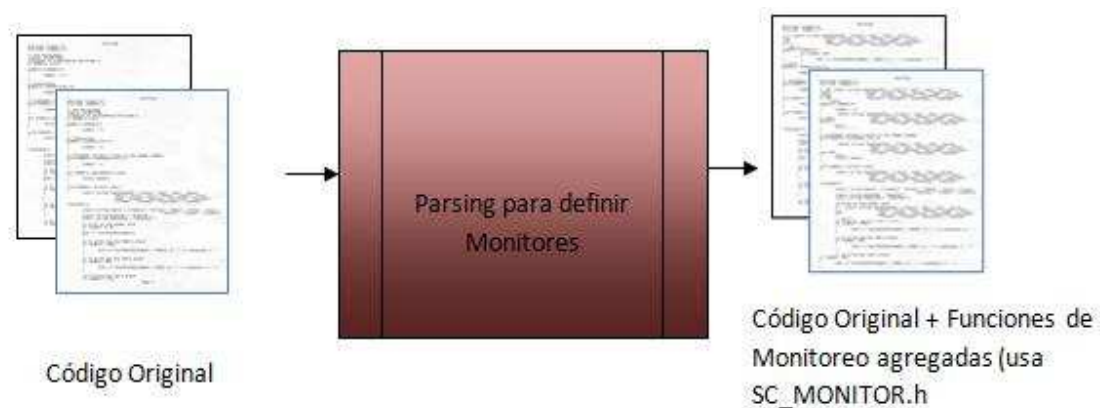
Para VHDL, el traductor dentro del VSP Compiler resultó ser el mejor, aunque es una herramienta comercial (se corrió un trial). Otras opciones son VH2SC o VHDL-to-SystemC-Converter de [www.ti.informatik.uni](http://www.ti.informatik.uni) son sin costo.

### PRE-SIMULACIÓN.

Se ha elaborado una biblioteca de monitores (SC\_MONITOR.h) orientados a los tipos de datos usados en SystemC para diseño con el fin de descubrir aquellos bloques dentro de un diseño que presenten una suficiente cantidad de transacciones para poder ser particionados. La métrica para decidir cuál es la cantidad suficiente para particionar viene dada por un estudio estadístico de las transacciones que ocurren en el diseño basándose en un plan de verificación y puntos de cobertura.



El proceso de monitoreo pretende descubrir interfaces de mínima actividad que permita optimizar el particionamiento de los bloques donde se realizan altos niveles de transacciones. De este modo, la segunda etapa consiste en hacer un parsing del código para identificar todos los nodos (señales de entrada, de salida, de procesamiento, buffers, etc.) de los cuales se vale el sistema para procesar sus datos, (Véase Figura 4.15). Después de haberlos identificado, hay que colocar estas “puntas” de prueba sobre ellos de tal forma que midan en una presimulación la cantidad de transacciones que manejan todos y cada uno de esos nodos.



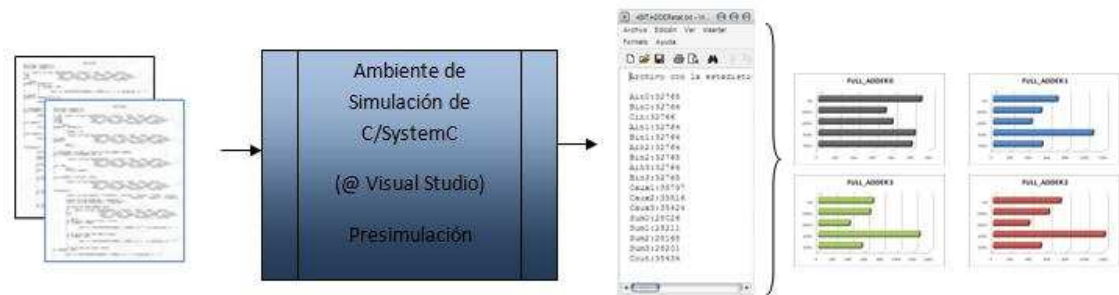
**Figura 4.15.** Parsing al código original.

[45] nos da una referencia confiable por la cual corriendo una simulación breve del diseño podemos estimar la carga computacional asociada con cada nodo. Durante esta pre simulación, un contador es mantenido para cada nodo del sistema, y cada vez que en alguno de ellos se detecta un cambio, el contador es incrementado.

Para las pruebas hechas, las medidas obtenidas en la evaluación funcional durante solo un 10% del total de la simulación resultaron consistentes con el comportamiento global de los nodos. Esto indica que nuestro algoritmo de particionamiento posterior puede descansar razonablemente en una pre simulación que permita predecir las cargas de trabajo que tendrá cada nodo (Véase Figura 4.16).

Al final, la pre simulación arroja, un vector en un archivo de texto y con ello los nombres de los nodos y la cantidad de transacciones hechas por cada uno de ellos.





**Figura 4.16.** Pre simulación para la predicción de las cargas de trabajo en los nodos.

### GRAFO DEL SISTEMA.

El objetivo del particionador en nuestra plataforma es proveer una cama de pruebas versátil y escalable con dos finalidades: probar diferentes algoritmos de particionamiento y aplicarlos a diferentes implementaciones de un circuito. Con esto en mente, ideamos un particionador consistente en dos bloques: el grafo del circuito a particionar y el algoritmo de particionamiento a usar.

Sin necesidad de pasar por la etapa anterior, en este bloque obtenemos el grafo que describe al sistema primero de forma abstracta (parecido a una matriz de adyacencia) y después de forma grafica para el mejor entendimiento del sistema a ser particionado.

En esta fase, se utiliza la función del parser anterior para encontrar los nodos pero también se implementaron funciones que detectan las relaciones entre ellos, lo que dará lugar a los arcos o conexiones entre ellas en el grafo (Véase Figura 4.17).

La forma abstracta que describe a este grafo es una tabla que primero lista los nodos y después enlista cada arco, uno en cada línea, para dar lugar a las conexiones.

Opcionalmente, el programa puede leer el archivo de estadística para agregar los números como pesos en cada arco según corresponda.



## CAPITULO 4 Pruebas y Resultados



```
directed,weighted,([A(e:8,8,50,50,n:Ain0,c:255,63,000),
B(e:145,8,50,50,n:Bin0,c:255,63,000),
C(e:283,8,50,50,n:Cin,c:255,63,000),
D(e:420,8,50,50,n:Ain1,c:255,63,000),
E(e:558,8,50,50,n:Bin1,c:255,63,000),
F(e:695,8,50,50,n:Ain2,c:255,63,000),
G(e:833,8,50,50,n:Bin2,c:255,63,000),
H(e:970,8,50,50,n:Ain3,c:255,63,000),
I(e:1108,8,50,50,n:Bin3,c:255,63,000),
J(e:8,146,100,100,n:FA0,c:255,255,255),
K(e:191,371,50,50,n:Caux1,c:255,255,255),
L(e:374,546,100,100,n:FA1,c:255,255,255),
M(e:558,371,50,50,n:Caux2,c:255,255,255),
N(e:741,146,100,100,n:FA2,c:255,255,255),
O(e:924,371,50,50,n:Caux3,c:255,255,255),
P(e:1108,546,100,100,n:FA3,c:255,255,255),
Q(e:8,792,50,50,n:Sum0,c:000,127,255),
R(e:283,792,50,50,n:Sum1,c:000,127,255),
S(e:558,792,50,50,n:Sum2,c:000,127,255),
T(e:833,792,50,50,n:Sum3,c:000,127,255),
U(e:1108,792,50,50,n:Cout,c:000,127,255)],[{A,J,d:1,v:32764},
{B,J,d:1,v:32766},
{C,J,d:1,v:32766},
{J,Q,d:1,v:28026},
{J,K,d:1,v:35797},
{D,L,d:1,v:32764},
{E,L,d:1,v:32764},
{K,L,d:1,v:35797},
{L,R,d:1,v:28211},
{L,M,d:1,v:35816},
{F,N,d:1,v:32764},
{G,N,d:1,v:32765},
{M,N,d:1,v:35816},
{N,S,d:1,v:28168},
{N,O,d:1,v:35424},
{H,F,d:1,v:32764},
{I,F,d:1,v:32765},
{O,F,d:1,v:35424},
{E,T,d:1,v:28201},
{E,U,d:1,v:35434},
```

Figura 4.17. Tabla de descripción del grafo.

Para visualizar esta tabla de forma grafica, nos valemos del Applet de CPMP Tools llamado Vertex-Edge. Disponible en <http://www.wmich.edu/cmp/>. Vista del Grafo: Figura 4.5.

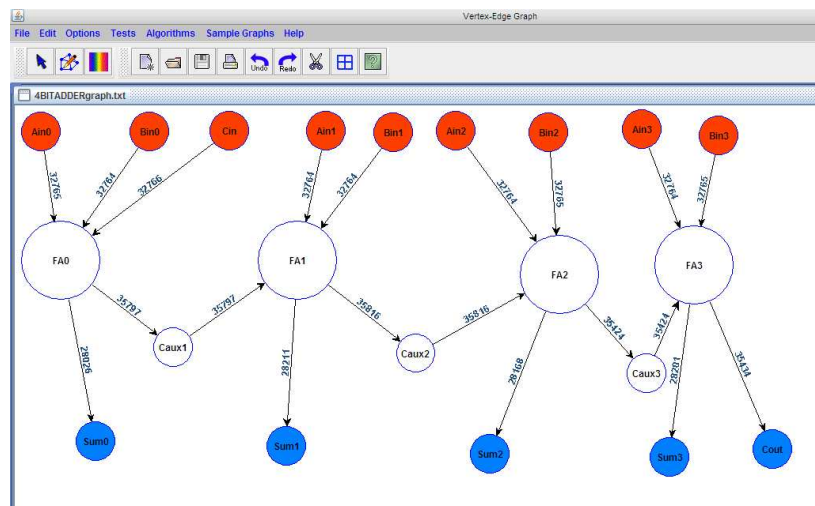


Figura 4.18. Construcción del grafo en forma grafica.



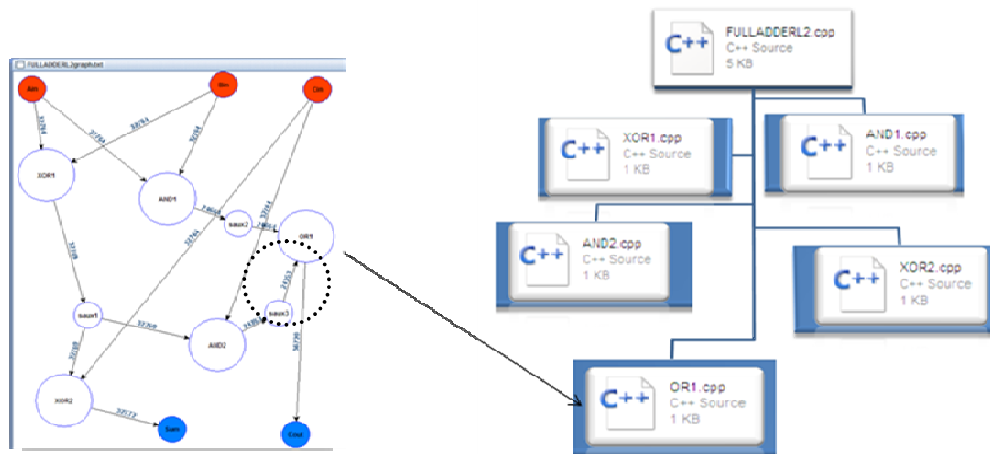
### PARTICIONAMIENTO.

Ahora bien, ya que tenemos un enfoque de la actividad efectiva del sistema a verificar, debemos formar las particiones que nos permitirán distribuir la simulación.

Aunque todas las etapas anteriores son importantes, esta es la crítica, pues aquí se harán los cortes y se crearán los módulos ejecutables por separado para su posterior distribución.

En el programa que se encarga de esta parte, se ha implementado un algoritmo que identifica los submódulos de los que está compuesto el sistema y los considera como una partición. Para esto, debe conocer las entradas y salidas de cada uno de estos submódulos, saber que tipos de dato maneja y además a cuáles les debe entregar información.

Este programa “desensambla” en submódulos independientes pero que se comunican entre sí mediante colas. La información obtenida por esta etapa de descomposición nos ayudará a “reconstruir” el diseño mapeando cada submódulo a una entidad de ejecución por separado, con la idea de que se ejecuten concurrentemente. (Véase Figura 4.19).



**Figura 4.19.** Creación de submódulos.

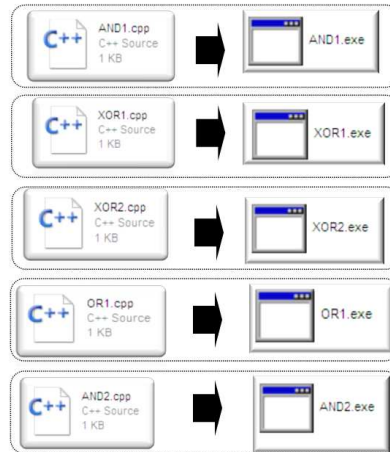
Después de esta etapa, un Script se encargará de hacer una compilación de los archivos en C++ usando la biblioteca de SystemC para todas y cada una de las particiones, (Véase Figura 4.20). Esto generará varios archivos ejecutables (el mismo número de particiones) teniendo así listos los elementos que serán distribuidos por el motor de ejecución paralela.



## CAPITULO 4 Pruebas y Resultados



La paralelización de estos procesos es lo que dará lugar a la aceleración en los tiempos de simulación, sin embargo, como pasa en cualquier programa distribuido, la sincronización es fundamental para asegurar un correcto procesamiento de los datos.



**Figura 4.20.** Compilación de los archivos en C++.

### 4.2.3 CARACTERÍSTICAS DEL APLICACIÓN UTILIZADA EN LAS PRUEBAS

En la Tabla 4.3 muestra la descripción de la aplicación utilizada para una de las pruebas realizadas al sistema, además se especifica el número de operaciones realizadas por este.

**Tabla 4.3.** Características de la aplicación.

NOMBRE	CANTIDAD DE OPERACIONES	DESCRIPCIÓN
1.- Sumador de 1 bit con acarreo.	Vectores de prueba * número de vértices. $(2^{10}) * 8 = 8192$	Representa la simulación de un circuito digital (Sumador de 3 bits) el cual trabaja con vectores de prueba de $2^{10}$ . Consta de 8 archivos ejecutables que interactúan entre sí para cumplir con su objetivo. Los nombres de los archivos son los siguientes: AIN, BIN, CIN, XOR1, XOR2, AND1, AND2, OR.



### 4.2.4 EVALUACIÓN DEL RENDIMIENTO DEL SISTEMA

La distribución de los vértices se puede observar en la Tabla 4.4. La distribución de procesos se realizo de manera adecuada, ya que los 6 trabajadores se clasificaron con alto poder de cómputo por lo cual pueden ejecutar cualquier proceso, además de ello el sistema siempre fue asignando un nuevo proceso al trabajador que menos tenía en ejecución.

**Tabla 4.4.** Distribución de los procesos entre los trabajadores.

NÚMERO DE TRABAJADORES	DISTRIBUCIÓN
1	148.204.64.205: AIN, BIN, CIN, XOR1, XOR2, AND1,AND2, OR1
2	148.204.64.205: BIN, XOR1, XOR2, OR1 148.204.64.209: AIN, CIN, AND1, AND2
3	148.204.64.205: BIN, AND1, OR1 148.204.64.209: AIN, XOR1, AND2 148.204.64.206: CIN, XOR2
4	148.204.64.205: CIN, AND2 148.204.64.209: AIN, AND1 148.204.64.206: OR1, XOR1 148.204.64.203: BIN, XOR2
5	148.204.64.205: XOR1 148.204.64.209: BIN, AND2 148.204.64.206: AND1 148.204.64.203: CIN, OR1 148.204.64.197: AIN, XOR2
6	148.204.64.205: XOR1 148.204.64.209: BIN, OR1 148.204.64.206: XOR2 148.204.64.203: CIN 148.204.64.197: AIN, AND2 148.204.64.199: AND1

En la Figura 4.21 se puede observar el rendimiento que tuvo el sistema para la simulación del sumador de 1 bits con acarreo, el cual fue particionado en 8 procesos; en dicha simulación participaron 6 trabajadores. La gráfica nos muestra que el tiempo de duración de la prueba incrementa entre más trabajadores haya, esto se debe a que la latencia de las comunicaciones entre trabajadores se vuelve mayor al tiempo que tarda cada proceso en realizar sus tareas. Esto se da



debido a que es más pequeño el tiempo de cómputo en comparación con el tiempo existente en las comunicaciones más el tiempo de lectura y escritura de las colas.



**Figura 4.21.** Gráfica que muestra el rendimiento del sistema.



# Capítulo 5

## *Conclusiones y Trabajos a Futuro*

---

**S**e muestran las conclusiones generadas para este trabajo de investigación además de que se describen los puntos que han quedado como trabajo a futuro.

El contenido de los temas presentados en este capítulo incluye:

- Conclusiones.
- Trabajos a futuro.



### 5.1.CONCLUSIONES

Se desarrolló una interfaz de usuario con la cual los usuarios pueden observar los resultados, eventos registrados en la bitácora, de las simulaciones que realizan. Además de ello se encarga de enviar todos aquellos procesos pertenecientes a dicha simulación al maestro por medio de la comunicación de objetos remotos.

Se desarrolló un esquema cliente/servidor llamado maestro, el cual recibe todos aquellos procesos pertenecientes a una simulación de algún circuito digital; encargándose también de la distribución de procesos, planificación y balanceo de las cargas de trabajo.

Se implemento el proceso de comunicación a través de objetos remotos, entre la interfaz de usuario y el maestro, logrando con ello hacer uso de objetos remotos para acceder a los métodos que pertenecen a las clases remotas en el servidor maestro. De esta manera se logró el envío de datos.

Se creó un proceso encargado de verificar los recursos de cada uno de los trabajadores como son: RAM libre, velocidad de la RAM, espacio en disco duro libre, velocidad y número de microprocesadores. De esta manera se puede verificar el estado de cada trabajador antes de enviarle más trabajo ayudando con ello a realizar una mejor planificación.

Se desarrolló un cliente/servidor llamado trabajador (puede ser copiado a todos los trabajadores pertenecientes a la Grid), el cual cumple con la tarea de ejecutar cada proceso enviado por el maestro, realizando el envío de resultados a otros trabajadores o a los usuarios.

Se implemento el proceso de comunicación a través de objetos remotos, entre el maestro y los trabajadores, haciendo uso de objetos remotos para acceder a los métodos que pertenecen a las clases remotas en el servidor de cada trabajador. De esta manera se envían los procesos, junto con la información que indica a donde se tienen que transportar los resultados.

Se implemento el proceso de comunicación en forma peer to peer entre trabajadores y usuarios, nuevamente haciendo uso de objetos remotos para acceder a los métodos que pertenecen a las clases remotas en el servidor de cada trabajador o de cada usuario logrando con ello dar soporte a las múltiples tareas con relaciones entre las dependencias de datos.

El sistema en términos de escalabilidad nos permite agregar nuevas computadoras a la Grid, tomando en cuenta que una de las principales ventajas de este sistema es que tiene la capacidad de trabajar con computadoras heterogéneas.

En cuanto a la forma de almacenamiento de resultados por medio de archivos de texto (colas) se demostró que funcionan de forma correcta. Ya que le permite al sistema almacenar dichos resultados de manera que siempre se vayan procesando en el orden correcto.





El rendimiento que se obtuvo para la simulación del sumador de 3 bits, basados en las pruebas realizadas, nos indican que a mayor número de trabajadores, el tiempo que tarda la simulación en terminar es mayor. Esto se debe a que la cantidad de cómputo que realiza cada proceso es considerablemente muy pequeña, lo cual deja muy a la vista que a mayor número de comunicaciones entre trabajadores crece la latencia de tal manera que al sistema le toma más tiempo en terminar dicha simulación. Se creó que si los procesos generarán una mayor cantidad de cómputo se podrá esconder la latencia de las comunicaciones entre los trabajadores, logrando con ello una disminución en el tiempo de las simulaciones entre mayor número de trabajadores exista.

### 5.2. TRABAJOS A FUTURO

Como trabajo a futuro quedan los siguientes puntos que ayudarían al sistema a tener mayor funcionalidad:

- La creación de nuevos modelos de planificación, distribución y balanceo de cargas de trabajo, se plantea que tales modelos podrían desarrollar técnicas inteligentes para fortalecer el performance y la funcionalidad del sistema.
- Encontrar mejores mecanismos de almacenamiento de resultados, por ejemplo a través de estructuras de memoria, ya que esto agilizaría la lectura de los resultados.
- La identificación y eliminación de cuellos de botella.
- Brindarle robustez al sistema haciéndolo tolerante a fallos, además de que tenga la capacidad de identificar donde se ha producido un error, recuperarse de este y continuar con la ejecución de la simulación.
- Unificación del sistema Grid Computing con el sistema de creación de vértices.



## REFERENCIAS BIBLIOGRÁFICAS

- [1] Educación virtual ¿Una alternativa a la educación tradicional?, Obtenida el día 09 de Junio del 2009, <http://www.utdt.edu/eduforum/ensayo10.htm>
- [2] L. Rosado, J.R. Herreros, (2009) “Nuevas aportaciones didácticas de los laboratorios virtuales y remotos en la enseñanza de la Física”, Conferencia Internacional sobre multimedia y tecnologías de la comunicación y la información en la educación, Lisbon Portugal.
- [3] Lijun Li, Hai Huang and Carl Tropper, (2003) “DVS: An Object-Oriented Framework for Distributed Verilog Simulation”, Procedente de la conferencia simulación paralela y distribuida
- [4] Francis L. Chan, Mark D. Spiller, A. Richard Newton, (1998) “ WELD – An environment for Web-Based Electronic Design”, Procedente de la 35 conferencia anual sobre automatización del diseño (San Francisco, California, Estados Unidos).
- [5] B. Balamuralithara, (2007) “Virtual Laboratories in Engineering Education: The Simulation Lab and Remote Lab”, Cyberjaya, Malaysia.
- [6] Wikipedia, Historia del Grid Computing, Obtenida el 2 de Marzo de 2009, [http://en.wikipedia.org/wiki/Grid\\_computing#History](http://en.wikipedia.org/wiki/Grid_computing#History).
- [7] Chao Jin, Rajkumar Buyya, (2007) “A dataflow model for .NET-based Grid Computing systems”, Procedente del laboratorio de Grid Computing y sistemas distribuidos, departamento de informática e ingeniería de software de la Universidad (Melbourn, Australia).
- [8] *Xingchen Chu, Krishna Nadiminti, Chao Jin, Srikumar Venugopal, Rajkumar Buyya*,(2007), “Aneka: Next-Generation Enterprise Grid Platform for e-Science and e-Business Applications”, Procedente de la tercera conferencia internacional IEEE sobre e-Ciencia y Grid Computing.
- [9] Akshay Luther, Rajkumar Buyya, Rajiv Ranjan, and Srikumar Venugopal, (2005) “Alchemi: A .NET-based Enterprise Grid Computing System”, Procedente del laboratorio de Grid Computing y sistemas distribuidos, departamento de informática e ingeniería de software de la Universidad (Melbourn, Australia).
- [10] Tiberiu STEF-PRAUN, Benjamin CLIFFORD, Ian FOSTER, Uri HASSON, Mihael Hategan, Steven L. Small , Michael Wilde, Yong Zhao, (2007) “ Accelerating Medical Research using the Swift Workflow System”, Procedente de la Universidad de Chicago, USA.
- [11] Edward Ashford Lee, And Daving G. Messerschmitt, (1987), “Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing”.



- [12] Q. Peng D.P. Schissel M. Thompson I. Foster M. Greenwald D. McCune K. Keahey, T. Fredian, (2002), “The Anatomy of the Grid: Enabling Scalable Virtual Organizations”, Procedente del primer IEEE/ACM simposio Computación de clúster y Grids.
- [13] J.Ñick S. Tuecke I. Foster, C. Kesselman, (2002), “The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration”, Procedente del foro global sobre Grid, USA.
- [14] T. Durniak P. Herman J. Karuturi C.Woods C. Gilman J. Barry, M. Aparicio, (1998) “Enterprise Distributed Computing Workshop”, Procedente de IEEE, USA.
- [15] Jay Unger, (2003),” The Physiology of the Grid: A visual of Open Grid Services Architecture”, IBM.
- [16] L. Joyanes Aguilar, (1997), Cibersociedad. Mac Graw-Hill, Interamericana de España, S.A.
- [17] H.Ñimrod D. Abramson, R. Giddy Susic, (1995), “The Grid: Blueprint for a New Computing Infrastructure”, Procedente de IEEE, USA.
- [18] E. Freeman, S. Hupfer, and K. Arnold, (1999), “*JavaSpaces: Principles, Patterns, and Practice*”, Addison-Wesley.
- [19] C. Lee, E. Coe, B. Michel, J. Clark, and B. Davis, (2002), “*Using advanced communication services in grid environments*”. Procedente del cuarto taller internacional sobre servicios activos de middleware.
- [20] Message Passing Interface Forum, (1995), “*MPI: A Message Passing Interface Standard*”, [www.mpi-forum.org](http://www.mpi-forum.org), Procedente de la Universidad de Tennessee, USA.
- [21] Message Passing Interface Forum, (1997), “*MPI-2: Extensions to the Message Passing Interface*”, Procedente de la Universidad de Tennessee, USA.
- [22] H. Nakada, S. Matsuoka, K. Seymour, J. Dongarra, C. Lee, and H. Casanova. “*GridRPC: A Remote Procedure Call API for Grid Computing*”, Procedente del reporte técnico, Universidad de Tennessee, USA.
- [23] V. Getov, G. von Laszewski, M. Philippsen, and I. Foster, (2001), “*Multi-Paradigm Communications in Java for Grid Computing*”, Procedente de comunicaciones de ACM, páginas 118–125, USA.
- [24] Blaise Barney, (1997), “*OpenMP C and C++ Application Program Interface*”, [www.openmp.org](http://www.openmp.org).



- [25] M. Sato, M. Hirono, Y. Tanaka, and S. Sekiguchi, (2001), “*OmniRPC: A Grid RPC Facility for Cluster and Global Computing in OpenMP*”, Procedente de *WOMPAT, LNCS 2104*, páginas 130– 136, Springer-Verlag.
- [26] B. Carpenter et al, (2000.), “*MPJ: MPI-like Message-passing for Java. Concurrency: Practice and Experience*”, páginas 1019–1038.
- [27] L. Gong, (2002), “*Special Issue on Peer-to-Peer Networking. IEEE*”, Editor. IEEE computo en Internet, volumen 6(1).
- [28] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, (2002), “*Grid Services for Distributed System*”, Procedente de integración de computadoras IEEE, páginas 37–46.
- [29] D. Marinescu and C. Lee, ( 2002), “ *Process Coordination and Ubiquitous Computing*”, CRC Press, Boca Raton, FL, USA.
- [30] M. Beynon, T. Kurc, A. Sussman, and J. Saltz, (2001) “*Optimizing execution of component-based applications using group instances*”. Procedente del simposio internacional IEEE sobre computo en clúster la Grid (*CCGrid 2001*), Brisbane, Australia.
- [31] M. D. Beynon, R. Ferreira, T. Kurc, A. Sussman, and J. Saltz, (2000), “*DataCutter: Middleware for filtering very large scientific datasets on archival storage systems*”. Procedente de *MASS2000*, páginas 119–133, Aeronáutica Nacional y espacios de administración, NASA/CP 2000-209888.
- [32] H. Zima and T. Sterling, (2000), “*Macroservers: An objectbased programming and execution model for processor-inmemory arrays*”. Procedente de simposio internacional sobrealto funcionamiento en el computo (*ISHPC2K*).
- [33] S. Baden and S. Fink, (1999), “*The Data Mover: A machine independent abstraction for managing customized data motion. LCPC*”.
- [34] Microsoft Corporation, Información general del framework de .NET Remoting, Obtenida el 27 de Marzo de 2009, [http://msdn.microsoft.com/es-es/library/kwtd6w2k\(VS.80\).aspx](http://msdn.microsoft.com/es-es/library/kwtd6w2k(VS.80).aspx)
- [35] Microsoft Corporation, Arquitectura del framework .NET Remoting, Obtenida el 27 de Marzo de 2009, [http://msdn.microsoft.com/es-es/library/2e7z38xb\(VS.80\).aspx](http://msdn.microsoft.com/es-es/library/2e7z38xb(VS.80).aspx)
- [36] Microsoft Corporation , Procesos y Dominios de Aplicación, Obtenida el 27 de Marzo de 2009, [http://msdn.microsoft.com/es-es/library/kt21t9h7\(VS.80\).aspx](http://msdn.microsoft.com/es-es/library/kt21t9h7(VS.80).aspx)
- [37] Microsoft Corporation, Objetos utilizables y no utilizables de forma remota, Obtenida el 27 de Marzo de 2009, [http://msdn.microsoft.com/es-es/library/h8f0y3fc\(VS.80\).aspx](http://msdn.microsoft.com/es-es/library/h8f0y3fc(VS.80).aspx)



- [38] Microsoft Corporation , Periodos y activación de objetos, Obtenida el 27 de Marzo de 2009, [http://msdn.microsoft.com/es-es/library/9ze044wd\(VS.80\).aspx](http://msdn.microsoft.com/es-es/library/9ze044wd(VS.80).aspx)
- [39] Microsoft Corporation, Canales, Obtenida el 27 de Marzo de 2009, [http://msdn.microsoft.com/es-es/library/dkfd3wha\(VS.80\).aspx](http://msdn.microsoft.com/es-es/library/dkfd3wha(VS.80).aspx)
- [40] Modelo de clases, Obtenida el 15 de Abril de 2009, <http://www.dcc.uchile.cl/~psalinas/uml/modelo.html>
- [41] Harvey M. Deitel y Paul J. Deitel, (2007), “C# Como Programar”, segunda edición, Pearson, páginas 170 – 171.
- [42] Programación con C# y .NET Remoting / Mono Remoting, Obtenida el 17 de Abril de 2009, <http://www.eridem.net/programacion-con-c-y-net-remoting-mono-remoting-vi/>
- [43] Programación con C# y .NET Remoting / Mono Remoting, Obtenida el 17 de Abril de 2009, <http://www.eridem.net/programacion-con-c-y-net-remoting-mono-remoting-vii/#elPorQue>
- [44] Comunicación Peer to Peer, Obtenida el 22 de Abril de 2009, [http://es.wikipedia.org/wiki/Caracteristicas\\_p2p](http://es.wikipedia.org/wiki/Caracteristicas_p2p).
- [45] Roger D. Chamberlain & Cheryl Henderson. “Evaluating the use of Pre-Simulation in VLSI Circuit Partitioning”.
- [46] Francis L. Chan, Mark D. Spiller, A. Richard Newton, (1998), “WELD—an environment for Web-based electronic design”, Procedente de la Universidad de San Francisco California, Estados Unidos.