7. CONCLUSIONES Y TRABAJO FUTURO

Debido al rápido avance de la tecnología y a la necesidad de utilizar los procesadores de los equipos existentes, las diferentes comunidades científicas se vieron en la tarea de utilizar cluster heterogéneos, sin embargo, conectar cada uno de los nodos con recursos de tipos y/o capacidades distintas puede ayudar al bajo desempeño, sí las diferencias en el poder de procesamiento entre los nodos no son observadas y tratadas de forma correcta.

Un cluster destinado a aplicaciones paralelas de alto desempeño y formado por nodos heterogéneos, puede presentar bajo desempeño si todos los nodos son tratados indiscriminadamente. Este bajo desempeño tiende a ser más expresivo cuanto mayor sea la diferencia entre el poder computacional de los nodos. Buscando aumentar el desempeño en el cluster, son aplicadas algunas técnicas de balanceo de carga, en las que se hace que los nodos reciban carga de acuerdo a sus propias capacidades en cuanto a recurso computacional se refiere.

En este trabajo se enfatizó en que cada nodo sólo pudiera efectuar trabajos que no excedieran sus capacidades reales, con el fin de mejorar los recursos del cluster, adoptando cargas grandes sólo por procesadores capaces de ejecutar dichas cargas. Así, el tiempo total de ejecución se logró minimizar en grandes porcentajes por el hecho de que cada nodo efectuaba lo que podía realizar sin exceder un límite.

Con respecto a los objetivos de esta investigación, se pudo apreciar que todos se cumplieron a cabalidad. Se logró hacer la correcta diferenciación entre los diversos tipos de cluster, escogiendo los de alto desempeño como propósito fundamental para el método planteado, además se pudo encaminar el método para el tipo cluster con el que se desee trabajar.

En cuanto al aspecto del balanceo de carga, está a escogencia de cada persona trabajar con la forma que mejor se adapte a sus necesidades. En esta investigación se trabajó con el balanceo de carga dinámico centralizado por el hecho de que el método se fundamenta en la distribución de la carga por parte de un maestro a sus esclavos repartiendo las tareas con base a las capacidades propias de cada nodo esclavo.

El método propuesto fue aplicado a un cluster heterogéneo trabajando con una de las arquitecturas que es netamente paralela: la arquitectura MIMD (Múltiples Instrucciones, Múltiples Datos), debido a que cada procesador es capaz de ejecutar un programa independiente de los demás procesadores, es decir, cada nodo esclavo es capaz de realizar las tareas asignadas por el nodo maestro independientemente de sí otro esclavo está o no ejecutando en el mismo instante de tiempo alguna otra tarea.

Finalmente, para la realización de la validación se tomaron tamaños de tareas conocidos con el fin de hacer comparables cada uno de los métodos, así, se logró probar que de manera general y recopilando todos los tiempos de ejecución totales para todas las pruebas

realizadas, el promedio del tiempo de ejecución total del método propuesto es mucho más efectivo con respecto a los otros dos métodos, mejorando en un 40.07% al método uniforme y en un 25.97% al método aleatorio.

Como trabajo futuro se propone implementar este método de balanceo de carga en sistemas reales para que éste pueda ser utilizado en grandes industrias y en tiempo real haciendo modificaciones en la topología de red, trabajando por ejemplo en topologías en estrella extendida donde se utilicen varios nodos maestros con grandes capacidades con el fin de que sean capaces de hacer la distribución de la carga de trabajo al resto de nodos esclavos de una manera más eficiente, trabajando así, con un tipo de arquitectura diferente a la de cluster de computadores y llegando a una arquitectura de computación grid.

Se propone también la realización de una evaluación en tiempo real para el proceso de identificación de los atributos definidos en la función de aptitud (velocidad de procesamiento, capacidad de memoria y latencia) en forma dinámica para cada nodo esclavo y la correcta definición de los pesos asociados a dichos atributos.

Por otro lado, también se propone realizar algún trabajo en el que se estudie el cómo hacer que la función de aptitud sea variable, implementando por ejemplo, un sistema experto basado en reglas con el fin de que no sólo sea un algoritmo de balanceo de carga enfatizado en rendimiento, sino también en confiabilidad y disponibilidad, o simplemente un algoritmo genérico basado en los requerimientos del usuario final.

8. REFERENCIAS

[ABAD, 2003]. ABAD, P., HERRERO, J. A., MENÉNDEZ DE LLANO, R., GARRIDO, S. & VALLEJO, F. “Análisis y evaluación de sistemas de colas en un entorno HTC”. XIV Jornadas de Paralelismo. Grupo de Arquitectura y Tecnología de Computadores. Departamento de Electrónica y Computadores. Universidad de Cantabria. Madrid, Septiembre de 2003.

[AKL, 1989]. AKL, SELIM G. “The design and analysis of parallel algorithms”. ISBN:0-13-200056-3. Prentice Hall Englewood Cliffs, NJ. 1989.

[ANDRESEN, 2003]. ANDRESEN, D., SCHOPF, N., BOWKER, E. & BOWER, T. “Distop: A low-overhead cluster monitoring system”. In Proceedings of the PDPTA. Las Vegas, pág. 1832 – 1836. Junio de 2003.

[ARAUJO, 2004]. ARAUJO CARDENAS, A. “Redes y sus topologías”. Junio de 2004. Disponible en: http://mx.geocities.com/alfonsoaraujocardenas/topologias.html

[ATTIYA, 2004]. ATTIYA, G. & HAMAM, Y. “Two phase algorithm for load balancing in heterogeneous distributed systems”. Proceedings of 12th Euromicro Conference Parallel, Distributed and Network-Based Processing. on 11-13, pág. 434 – 439. Febrero de 2004.

[AVERSA, 2000]. AVERSA, L. & BESTAVROS, A. “Load balancing a cluster of web servers using distributed packet rewriting”. IEEE Int’l Performance, Computing and Communication Conf. pág. 24 – 29. Febrero de 2000.

[AVRESKY, 2005]. AVRESKY, D. & NATCHEV, N. “Dynamic reconfiguration in computer clusters with irregular topologies in the presence of multiple node and link failures”. IEEE Transactions Computers. Volume 54, número 5, pág. 603 – 615, mayo de 2005.

[BARAK, 2009]. BARAK, A. & SHILOH, A. “The Mosix2 Management System for Linux Clusters and Multi-Clusters”. Mosix Org. Mayo de 2009.

[BEVILACQUA, 1999]. BEVILACQUA, A. “Dynamic load balancing method on a heterogeneous cluster of workstations”. Informatica. Volume 23, número 1, pág. 49 – 56. Marzo de 1999.

[BOHN, 1999]. BOHN C. A. & LAMONT G. B. “Asymmetric load balancing on a heterogeneous cluster of PCs”. In Proceedings of the PDPTA. Las Vegas, volume 5, pág. 2515 – 2522, 1999.

[BOONIC, 2008]. “Algoritmo paralelo”. Diciembre de 2008. Disponible en: http://www.boonic.com/enciclopedia/65306.php
http://es.wikipedia.org/wiki/Algoritmo_paralelo

[CARNS, 1999]. CARNS, P. H., LIGON III, W. B., MCMILLAN, S. P. & ROSS, R. B. “An Evaluation of Message Passing Implementations on Beowulf Workstations”. Proceedings of the 1999 IEEE Aerospace Conference. Marzo de 1999.

[CATALÁN, 2004]. CATALÁN I COIT, M. “El manual para el clustering con openMosix”. Versión Beta 1.0. Septiembre de 2004. Disponible en: http://es.tldp.org/Manuales-LuCAS/doc-manual-openMosix-1.0/doc-manual-openMosix-1.0.pdf

[CATALÁN, 2006]. CATALÁN, M. “El manual para el clustering con openMosix. Teoría de la supercomputación”. Traducción del trabajo realizado por Kris Buytaert HOWTO escrito en Febrero de 2002. Disponible en: http://www.wikilearning.com/tutorial/el_manual_para_el_clustering_con_openmosix-open_mosix/9756-1

[CISCO SYSTEMS, INC, 2009]. Academia Regional Universidad Nacional de Colombia. Cisco Networking Academy Program. Disponible en: http://rei.unalmed.edu.co/

[CUETO, 2004]. CUETO DÍAZ, L. F. “Estudio e Implementación de un Cluster Clase Beowulf”. Pontificia Universidad Católica de Valparaiso (PUCV). Facultad de Ingeniería. Escuela de Ingeniería Eléctrica (EIE). Chile.

[CHAU, 2003]. CHAU, S. C. & FU, A. W. “Load balancing between computing clusters”. Proceedings of the Fourth International Conference. Parallel and Distributed Computing, Applications and Technologies, PDCAT'2003. pág. 548 – 551, 27-29. Agosto de 2003.

[CHIOLA, 1998]. CHIOLA, G. “Workshop F. Network Computing Keynote Speech Some Research Projects on Clusters of Personal Computers”. 24th. Proceedings of Euromicro Conference. Volume 2, pág. XLVII – XLLIV, de 25 a 27 de Agosto de 1998.

[CHIRINOV, 2003]. CHIRINOV, R. “Proyecto Cluster openMosix (Linux)”. 2003. Disponible en: http://www.noticias3d.com/articulo.asp?idarticulo=248&pag=4

[CHOI, 2003]. CHOI, M., YU, J., KIM, H. & MAENG, S. “Improving performance of a dynamic load balancing system by using number of effective tasks on Cluster Computing”. Proceedings of IEEE International Conference. pág. 436 – 441, 2003.

[DEBIAN, 2007]. “Debian”. Octubre de 2007. Disponible en: http://www.guia-ubuntu.org/index.php?title=Debian

[DECKER, 1997]. DECKER, T. “Virtual Data Space - A universal load balancing scheme”. In Proceedings of the 4-th International Symposium on Solving Irregularly Structured Problems in Parallel. Volume 1253 of Lecture Notes in Computer Science, pág. 159 – 166, 1997.

[DE GIUSTI, 2005]. DE GIUSTI, A., NAIOUF, M., DE GIUSTI L., CHICHIZOLA F., TINETTI, F., BARBIERI A., DENHAM M., VILLAGARCÍA H. & COSENTINO A. “Investigación en Sistemas Paralelos”. Instituto de Investigación en Informática LIDI. Facultad de Informática. Universidad Nacional de La Plata. Argentina. VII Workshop de Investigadores en Ciencias de la Computación. WICC 2005. Río Cuarto. Argentina. ISBN 950-665-337-2. Mayo de 2005.

[DORMIDO, 2003]. DORMIDO CANTO, S., HERNÁNDEZ BERRINCHES, R., ROS MUÑOZ, S. & SÁNCHEZ MORENO, J. “Procesamiento Paralelo. Teoría y Programación”. Editorial Sanz y Torres. Madrid, Enero de 2003.

[DROZDOWSKI, 2003]. DROZDOWSKI, M. & WOLNIEWICZ, P. “Out-of-core divisible load processing”. IEEE Transactions on Parallel and Distributed Systems, Volume 14, número 10, pág. 1048 – 1056. Octubre de 2003.

[ENGLER, 2004]. ENGLER, V. “Novedosa red informática para tratar problemas científicos complejos. La tecnología Grid desembarca en la Argentina”. Centro de Divulgación Científica - FCEyN. Septiembre de 2004. Disponible en: http://www.fcen.uba.ar/prensa/noticias/2004/noticias_06sep_2004.html

[EVIUXA, 2004]. EVIUXA. “Superordenadores”. Trabajo universitario. México. Disponible en: http://html.rincondelvago.com/superordenadores.html

[GARCÍA, 2003]. GARCÍA MERAYO, F. “La supercomputación”. Autores científico-técnicos y académicos (ACTA). Universidad Politécnica de Madrid. 2003. Disponible en: http://www.acta.es/articulos_mf/30009.pdf

[GEORGE, 1999]. GEORGE, W. “Dynamic load-balancing for data-parallel MPI programs”. National Institute of Standards and Technology. In Message Passing Interface Developer's and User's Conference (MPIDC'99). pág. 95 – 100, 1999.

[GIAR, 2007]. Grupo de Inteligencia Artificial y Robótica (GIAR). “Inteligencia Artificial”. Universidad Tecnológica Nacional. Facultad Regional Buenos Aires. Argentina, 2007. Disponible en: www.secyt.frba.utn.edu.ar/gia/inteligencia_artificial.htm

[GOTTLIEB, 1989]. GOTTLIEB, A. & ALMASI, G. S. “Highly Parallel Computing”. Benjamin-Cummings publishers, Redwood City, CA, 1989.

[HAASE, 2005]. HAASE, J., ESCHMANN, F. & WALDSCHMIDT, K. “The SDVM - An Approach for Future Adaptive Computer Clusters”. Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05). Washington, volume 17, pág. 278a – 278a, de 04 a 08 de abril de 2005.

[HERRERA, 2004]. HERRERA, J. M. “NS2 - Network Simulator”. Valparaíso. Mayo de 2004.

[HIDALGO, 2004]. HIDALGO PÉREZ, J. I. & CERVIGÓN RÜCKAUER, C. “Una revisión de los algoritmos evolutivos y sus aplicaciones”. Madrid, España. Disponible en: http://www.cesfelipesegundo.com/revista/Articulos2004b/Articulo5.pdf

[HOEGER, 2000]. HOEGER, H. “Introducción a la computación paralela”. Centro nacional de cálculo científico. Universidad de los Andes (CeCaLCULA). Mérida – Venezuela.

[HOGANSON, 1999]. HOGANSON, K. E. “Workload execution strategies and parallel speedup on clustered computers”. IEEE Transactions Computers. Volume 48, número 11, pág. 1173 – 1182. Noviembre de 1999.

[IBRAHIM, 2002]. IBRAHIM, M. A. M. & XINDA, L. “Performance of dynamic load balancing algorithm on cluster of workstations and PCs”. Shanghai Jiao Tong University, Computer Science and Engineering Department. Proceedings of Algorithms and Architectures for Parallel Processing. Fifth International Conference on 23-25, pág. 44 – 47. Octubre de 2002.

[KACER, 2002]. KACER, M. & TVRDÍK, P. “Load balancing by remote execution of short processes on linux clusters”. IEEE/ACM International Symposium on Cluster Computing and the Grid, 2002.

[KNOPPIX, 2009]. KNOPPIX. “Knoppix”. Febrero de 2009. Disponible en: http://es.wikipedia.org/wiki/Knoppix

[LANZA, 1999]. LANZA, C., HURWITZ, L., COLGAN, P., ALVAREZ, G. & SANTOS, J. J. “Auge y ocaso de la supercomputacion: la figura de Seymour Cray”. Lo que cuentan de ti. LQCTI N-9. Magazine independiente sobre nueva economía, tecnología e innovación. P 18-20. Septiembre de 1999.

[LEDESMA, 2007]. LEDESMA, G. P., AGUIRRE, M. A., ZULETA, M. F. & PAYARES, N. L. “Simulación de flujos de fluidos utilizando grillas de cómputo”. Universidad de San Buenaventura. Santa Fé de Bogotá, Colombia, 2007. Disponible en: http://www.acis.org.co/fileadmin/Revista_107/11.pdf

[LIARTE, 2007]. LIARTE LÓPEZ, M. R. “Estudio, implementación y evaluación de entornos de computación de alto rendimiento HTC”. Escuela Técnica Superior de

Ingeniería de Telecomunicación, Universidad Politécnica de Cartagena. España, Diciembre de 2007.

[LIAO, 1999]. LIAO, C. J. & CHUNG, Y.C. “Tree-based parallel load-balancing methods for solution-adaptative finite element graphs on distributed memory multicomputers”. IEEE Transactions on parallel and distributed systems, Vol. 10, No. 4, Abril de 1999.

[LINUX, 2006]. LINUX. “Distribuciones de Linux”. Disponible en: http://www.linux-es.org/distribuciones.php

[LINUXCENTRO, 2007]. LINUXCENTRO. Software libre y GNU/Linux. Disponible en: http://www.linuxcentro.net/linux/staticpages/index.php?page=CdS_LiNuX

[LIZÁRRAGA, 2002]. LIZÁRRAGA CELAYA, C. “Cluster de Linux”. Departamento de Física. Universidad de Sonora. México.2002. Disponible en: http://clusters.fisica.uson.mx/

[MARTÍNEZ, 2001]. MARTÍNEZ RESTREPO, L. M. & RAMOS RUEDA, C. A. “Procesamiento Paralelo en un Cluster Beowulf”. Universidad de Antioquia. 2001.

[MEREDITH, 2003]. MEREDITH, M., CARRIGAN, T. & BROCKMAN, J., CLONINGER, T., PRIVOZNIK, J. & WILLIAMS, J. “Exploring beowulf clusters”. Journal of Computing Sciences in Colleges. Volume 18, número 4, pág. 268–284. Abril de 2003.

[MID, 2006]. Sistema Madrid en Investigación y Desarrollo. “Inteligencia Artificial”. Ciencia y Sociedad. España, 2006. Disponible en: http://www.madrimasd.org/cienciaysociedad/ateneo/temascandentes/inteligenciaartifi/default.asp

[MOSIX, 2009]. MOSIX. “MOSIX: Cluster and Multi-Cluster Management”. Disponible en: http://www.mosix.org/index.html

[NGUYEN, 2001]. NGUYEN, V. A. & PIERRE, S. “Scalability of computer clusters”. Ecole Polytechnique de Montreal. Department of Electrical and Computer Engineering. Electrical and Computer Engineering. Canadian Conference on Volume 1, pág. 405 – 409, de 13 a 16 de mayo de 2001.

[NIEeS, 2006]. The National Institute for Environmental eScience (NIEeS). “Condor”. Octubre de 2006. Disponible en: http://gridinfo.niees.ac.uk/index.php/Condor

[PERERA, 1999]. PERERA DOMÍNGUEZ, M. “ENIAC, matemáticas y computación científica”. Universidad de Sevilla. ISSN 1138-8927, Vol. 2, Nº 3, 1999 , pags. 495-518. 1999. Disponible en: http://divulgamat.ehu.es/weborriak/historia/Gaceta/Historia23.pdf

[PÉREZ, 2003]. PÉREZ SÁNCHEZ, L. “BLIP. Programación paralela basada en la programación lógica con restricciones”. Facultad de matemática y computación. Universidad de la Habana. Junio de 2003. Disponible en: http://www.di.unipi.it/~perez/articles/blip.pdf

[PÉREZ, 2001]. PÉREZ, M. A. “Arquitecturas Paralelas”. 2001. Disponible en: http://www.dragones.org/Biblioteca/Articulos/ArquitecturaParalelas.pdf

[PINO, 2002]. PINO MORILLAS, O., ARROYO MORENO, R. F. & NIEVAS MUÑOZ, F, J. “Los Clusters como Plataforma de Procesamiento Paralelo”. 2002. Disponible en: http://usuarios.lycos.es/lacaraoculta/descargas/Clusters_definitivo.pdf

[PLAZA, 2004]. PLAZA NIETO, E. J. “Cluster Heterogéneo de Computadoras”. 2004. Disponible en: http://www.ii.uam.es/~fjgomez/CursoVerano/instal/cluster.pdf

[PROYECTO DE GRADO, 2006]. PROYECTO DE GRADO. “Clusters Alto Desempeño”. Laboratorio de Simulación Numérica de Flujos a Superficie Libre. Uruguay, 2006. Disponible en: http://www.technetworld.info/

[PUENTES, 2005]. PUENTES FERNÁNDEZ, S. “”. Grupo de Informática, Facultad de Biología. Universidad de la Habana. Abril de 2005. Disponible en: http://fbio.uh.cu/bioinfo/glosario.html

[REDHAT, 2009]. “Red Hat Linux”. Enero de 2009. Disponible en: http://es.wikipedia.org/wiki/Red_Hat_Linux

[REGO, 2006]. REGO GORINO, F. “Balanceamento de carga em clusters de alto desempenho: uma extensão para a LAM/MPI”. Universidade Estadual de Maringá. 2006.

[ROMO, 2003]. ROMO, M. “Procesamiento Paralelo”. Universidad Internacional del Ecuador. Boletín 5. 2003. Disponible en: http://www.internacional.edu.ec/academica/informatica/creatividad/uide-bits/uide-bits-05-2003.pdf

[SAVVAS, 2004]. SAVVAS, I. K. & KECHADI, M. “Dynamic task scheduling in computing cluster environments”. Parallel and Distributed Computing. Third International Symposiumon/Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks, 2004. Third International Workshop, pág. 372 – 379, 5 a 7 de julio de 2004.

[SMERLING, 2001]. SMERLING, L. & TSCHANZ, R. “Procesamiento paralelo: qué tener en cuenta para aprovecharlo. Conceptos y alternativas en Linux”. Universidad Tecnológica Nacional. Facultad Regional Santa Fe. 2001.

[SUN, 2007]. SUN Microsystems. SunSource.net, Open Source Innovation Starts Here. GridEngine. Disponible en: http://gridengine.sunsource.net/nonav/source http://gridengine.sunsource.net/nonav/source/browse/~checkout~/gridengine/doc/htmlman/manuals.html

[SUSE, 2009]. “Suse Linux”. Febrero de 2009. Disponible en: http://es.wikipedia.org/wiki/SUSE_Linux

[TUREGANO, 2002]. TUREGANO MOLINA, J. “Simulador NS-2”. Asociacion Linux Albacete. Marzo de 2006

[VELASCO, 2004]. VELASCO, F. & RAMOS, F. “Performance Evaluation of Quality of Service Aware Request Placement Techniques”. Mexican National Council of Science and Technology Grant. Centro de investigación y estudios avanzados del I.P.N. Guadalajara, México. Septiembre 30 de 2004.

[VERDEJO, 2009]. VERDEJO ÁLVAREZ, G. “PFC: Sistemas de Computación Masiva SunGrid”. Technical University of Catalonia, Departament de Llenguatges i Sistemes Informàtics. Enero de 2009 Disponible en: http://gabriel.verdejo.alvarez.googlepages.com/PFC_clusterscomputacion.pdf

9. ANEXOS

9.1 CONCEPTOS FUNDAMENTALES DE REDES TELEINFORMÁTICAS

De [CISCO SYSTEMS, INC, 2009], una red se define como un grupo dispositivos conectados unos a otros para comunicarse, transmitir datos entre ellos, y compartir recursos y servicios.

Estos dispositivos se clasifican en dos grandes grupos. El primer grupo está compuesto por los dispositivos de usuario final. Los dispositivos de usuario final incluyen los computadores, impresoras, scanners, y demás dispositivos que brindan servicios directamente al usuario. El segundo grupo está formado por los dispositivos de red. Los dispositivos de red son todos aquellos que conectan entre sí a los dispositivos de usuario final, posibilitando su intercomunicación [CISCO SYSTEMS, INC, 2009].

Los dispositivos de usuario final que conectan a los usuarios con la red también se conocen con el nombre de hosts. Estos dispositivos permiten a los usuarios compartir, crear y obtener información. Los dispositivos host pueden existir sin una red, pero sin la red las capacidades de los hosts se ven sumamente limitadas. Los dispositivos host están físicamente conectados con los medios de red mediante una tarjeta de interfaz de red (NIC).

Una NIC, o adaptador LAN, provee capacidades de comunicación en red desde y hacia un computador. En los sistemas computacionales de escritorio, es una tarjeta de circuito impreso que reside en una ranura en la tarjeta madre y provee una interfaz de conexión a los medios de red. La NIC se comunica con la red a través de una conexión serial y con el computador a través de una conexión paralela. La NIC utiliza una Petición de interrupción (IRQ), una dirección de E/S y espacio de memoria superior para funcionar con el sistema operativo. Un valor IRQ (petición de interrupción) es un número asignado por medio del cual donde el computador puede esperar que un dispositivo específico lo interrumpa cuando dicho dispositivo envía al computador señales acerca de su operación. Por ejemplo, cuando una impresora ha terminado de imprimir, envía una señal de interrupción al computador. La señal interrumpe momentáneamente al computador de manera que este pueda decidir que procesamiento realizar a continuación. Debido a que múltiples señales al computador en la misma línea de interrupción pueden no ser entendidas por el computador, se debe especificar un valor único para cada dispositivo y su camino al computador. Antes de la existencia de los dispositivos Plug-and-Play (PnP), los usuarios a menudo tenían que configurar manualmente los valores de la IRQ, o estar al tanto de ellas, cuando se añadía un nuevo dispositivo al computador [CISCO SYSTEMS, INC, 2009].

Al seleccionar una NIC, hay que tener en cuenta los siguientes factores:

- ✓ Protocolos: Ethernet, Token Ring o FDDI
- ✓ Tipos de medios: Cable de par trenzado, cable coaxial, inalámbrico o fibra óptica
- ✓ Tipo de bus de sistema: PCI o ISA

Los dispositivos de red son los que transportan los datos que deben transferirse entre dispositivos de usuario final. Los dispositivos de red proporcionan el tendido de las conexiones de cable, la concentración de conexiones, la conversión de los formatos de datos y la administración de transferencia de datos. Algunos ejemplos de dispositivos que ejecutan estas funciones son los repetidores, hubs, puentes, switches y routers.

Un repetidor es un dispositivo de red que se utiliza para regenerar una señal. Los repetidores regeneran señales analógicas o digitales que se distorsionan a causa de pérdidas en la transmisión producidas por la atenuación. Un repetidor no toma decisiones inteligentes acerca del envío de paquetes como lo hace un router o puente.

Los hubs concentran las conexiones. En otras palabras, permiten que la red trate un grupo de hosts como si fuera una sola unidad. Esto sucede de manera pasiva, sin interferir en la transmisión de datos. Los hubs activos no sólo concentran hosts, sino que además regeneran señales.

Los puentes convierten los formatos de transmisión de datos de la red además de realizar la administración básica de la transmisión de datos. Los puentes, tal como su nombre lo indica, proporcionan las conexiones entre LAN. Los puentes no sólo conectan las LAN, sino que además verifican los datos para determinar si les corresponde o no cruzar el puente. Esto aumenta la eficiencia de cada parte de la red.

Los switches de grupos de trabajo agregan inteligencia a la administración de transferencia de datos. No sólo son capaces de determinar si los datos deben permanecer o no en una LAN, sino que pueden transferir los datos únicamente a la conexión que necesita esos datos. Otra diferencia entre un puente y un switch es que un switch no convierte formatos de transmisión de datos.

Los routers poseen todas las capacidades indicadas arriba. Los routers pueden regenerar señales, concentrar múltiples conexiones, convertir formatos de transmisión de datos, y manejar transferencias de datos. También pueden conectarse a una WAN, lo que les permite conectar LAN que se encuentran separadas por grandes distancias. Ninguno de los demás dispositivos puede proporcionar este tipo de conexión.

9.1.1 TOPOLOGÍAS DE RED

La topología de red define la estructura de una red. Una parte de la definición topológica es la topología física, que es la disposición real de los cables o medios. La otra parte es la topología lógica, que define la forma en que los hosts acceden a los medios para enviar datos. Las topologías físicas más comúnmente usadas son las siguientes:

- ✓ Una topología de bus usa un solo cable backbone que debe terminarse en ambos extremos. Todos los hosts se conectan directamente a este backbone. Esta topología tiene todos sus nodos conectados directamente a un enlace y no tiene ninguna otra conexión entre nodos (figura 22). Físicamente cada host está conectado a un cable común, por lo que se pueden comunicar directamente, aunque la ruptura del cable hace que los hosts queden desconectados [ARAUJO, 2004].

  Permite que todos los dispositivos de la red puedan ver todas las señales de todos los demás dispositivos, lo que puede ser ventajoso si se desea que todos los dispositivos obtengan esta información. Sin embargo, puede representar una desventaja, ya que es común que se produzcan problemas de tráfico y colisiones, que se pueden disminuir segmentando la red en varias partes. Es la topología más común en pequeñas LAN, con hub o switch final en uno de los extremos.

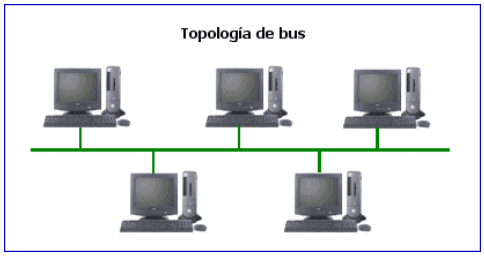

Figura 22. Topología de bus.
Tomada de [ARAUJO, 2004]

- ✓ La topología de anillo conecta un host con el siguiente y al último host con el primero. Esto crea un anillo físico de cable. Se compone de un solo anillo cerrado formado por nodos y enlaces, en el que cada nodo está conectado solamente con los dos nodos adyacentes. Los dispositivos se conectan directamente entre sí por medio de cables en lo que se denomina una cadena margarita (figura 23).
  Para que la información pueda circular, cada estación debe transferir la información a la estación adyacente [ARAUJO, 2004].

Figura 23. Topología en anillo.
Tomada de [ARAUJO, 2004]

- ✓ La topología de anillo doble consta de dos anillos concéntricos, donde cada host de la red está conectado a ambos anillos, aunque los dos anillos no están conectados directamente entre sí. Es análoga a la topología de anillo, con la diferencia de que, para incrementar la confiabilidad y flexibilidad de la red, hay un segundo anillo redundante que conecta los mismos dispositivos. La topología de anillo doble actúa como si fueran dos anillos independientes, de los cuales se usa solamente uno por vez [ARAUJO, 2004].

- ✓ La topología en estrella conecta todos los cables con un punto central de concentración. Tiene un nodo central desde el que se irradian todos los enlaces hacia los demás nodos (figura 24). Por el nodo central, generalmente ocupado por un hub, pasa toda la información que circula por la red. La ventaja principal es que permite que todos los nodos se comuniquen entre sí de manera conveniente. La desventaja principal es que si el nodo central falla, toda la red se desconecta [ARAUJO, 2004].

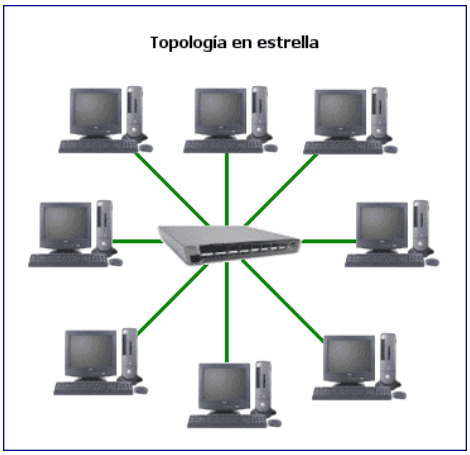

Figura 24. Topología en estrella.
Tomada de [ARAUJO, 2004]

- ✓ Una topología en estrella extendida conecta estrellas individuales entre sí mediante la conexión de hubs o switches. Esta topología puede extender el alcance y la cobertura de la red. Es igual a la topología en estrella, con la diferencia de que cada nodo que se conecta con el nodo central también es el centro de otra estrella. Generalmente el nodo central está ocupado por un hub o un switch, y los nodos secundarios por hubs. La ventaja de esto es que el cableado es más corto y limita la cantidad de dispositivos que se deben interconectar con cualquier nodo central. La topología en estrella extendida es sumamente jerárquica, y busca que la información se mantenga local. Esta es la forma de conexión utilizada actualmente por el sistema telefónico [ARAUJO, 2004].

- ✓ Una topología jerárquica es similar a una estrella extendida. Pero en lugar de conectar los hubs o switches entre sí, el sistema se conecta con un computador que controla el tráfico de la topología. Esta topología contiene un nodo de enlace troncal, el cual es por lo general un hub o switch, desde el que se ramifican los demás nodos (figura 25). El enlace troncal es un cable con varias capas de ramificaciones, y el flujo de información es jerárquico. Conectado en el otro extremo al enlace troncal generalmente se encuentra un host servidor [ARAUJO, 2004].

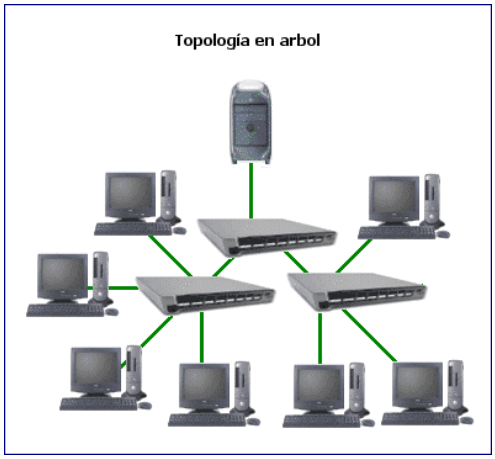

Figura 25. Topología en árbol
Tomada de [ARAUJO, 2004]

- ✓ La topología de malla se implementa para proporcionar la mayor protección posible con el fin de evitar una interrupción del servicio. El uso de una topología de malla en los sistemas de control en red de una planta nuclear sería un ejemplo excelente. Las ventajas de esta topología radican en que, como cada nodo se conecta físicamente a los demás, creando una conexión redundante, si algún enlace deja de funcionar la información puede circular a través de cualquier cantidad de enlaces hasta llegar a destino. Además, esta topología permite que la información circule por varias rutas a través de la red (figura 26). La desventaja física principal es que sólo funciona con una pequeña cantidad de nodos, ya que de lo contrario la cantidad de medios necesarios para los enlaces, y la cantidad de conexiones con los enlaces se torna abrumadora [ARAUJO, 2004].

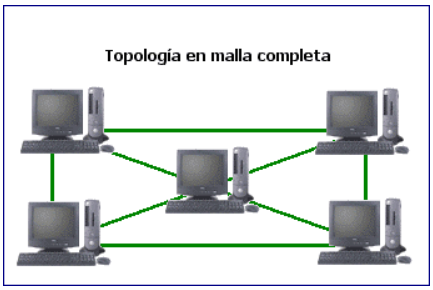

Figura 26. Topología en malla completa. Tomada de [ARAUJO, 2004]

- ✓ La topología de red celular está compuesta por áreas circulares o hexagonales, cada una de las cuales tiene un nodo individual en el centro (figura 27). La topología celular es un área geográfica dividida en regiones (celdas) para los fines de la tecnología inalámbrica. En esta tecnología no existen enlaces físicos; sólo hay ondas electromagnéticas. La ventaja obvia de una topología celular (inalámbrica) es que no existe ningún medio tangible aparte de la atmósfera terrestre o el del vacío del espacio exterior (y los satélites). Las desventajas son que las señales se encuentran presentes en cualquier lugar de la celda y, de ese modo, pueden sufrir disturbios y violaciones de seguridad. Como norma, las topologías basadas en celdas se integran con otras topologías, ya sea que usen la atmósfera o los satélites [ARAUJO, 2004].

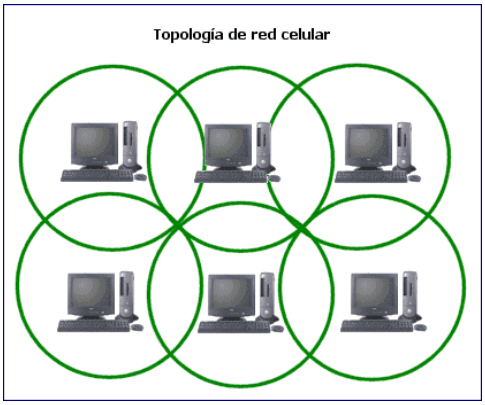

Figura 27. Topología de red celular. Tomada de [ARAUJO, 2004]

- ✓ La topología irregular es aquella para la cual no existe un patrón obvio de enlaces y nodos. El cableado no sigue un modelo determinado; de los nodos salen cantidades variables de cables. Las redes que se encuentran en las primeras etapas de construcción, o se encuentran mal planificadas, a menudo se conectan de esta manera [ARAUJO, 2004].

De acuerdo a [CISCO SYSTEMS, INC, 2009], la topología lógica de una red es la forma en que los hosts se comunican a través del medio. Los dos tipos más comunes de topologías lógicas son broadcast y transmisión de tokens.

- ✓ La topología broadcast simplemente significa que cada host envía sus datos hacia todos los demás hosts del medio de red. No existe una orden que las estaciones deban seguir para utilizar la red. Es por orden de llegada. Ethernet funciona así.

- ✓ La transmisión de tokens controla el acceso a la red mediante la transmisión de un token electrónico a cada host de forma secuencial. Cuando un host recibe el token, ese host puede enviar datos a través de la red. Si el host no tiene ningún dato para enviar, transmite el token al siguiente host y el proceso se vuelve a repetir.

9.1.2 TIPOS DE REDES DE COMPUTADORES

Atendiendo al ámbito que abarcan las telecomunicaciones, tradicionalmente se habla de:

- ✓ Redes de Área Local (LAN): conecta varias estaciones dentro de la misma institución. Las LAN constan de los siguientes componentes:
  - Computadores
  - Tarjetas de interfaz de red
  - Dispositivos periféricos
  - Medios de networking
  - Dispositivos de networking

  Las LAN permiten a las empresas aplicar tecnología informática para compartir localmente archivos e impresoras de manera eficiente, y posibilitar las comunicaciones internas [CISCO SYSTEMS, INC, 2009].

  Algunas de las tecnologías comunes de LAN son:
  - Ethernet: topología de bus lógica y en estrella física o en estrella extendida.
  - Token Ring: topología de anillo lógica y una topología física en estrella.
  - FDDI: topología de anillo lógica y topología física de anillo doble.

- ✓ Redes de Área Metropolitana (MAN): es una red que abarca un área metropolitana, como, por ejemplo, una ciudad o una zona suburbana. Una MAN generalmente consta de una o más LAN dentro de un área geográfica común. Por ejemplo, un banco con varias sucursales puede utilizar una MAN. Normalmente, se utiliza un proveedor de servicios para conectar dos o más sitios LAN utilizando líneas privadas de comunicación o servicios ópticos. También se puede crear una MAN usando tecnologías de puente inalámbrico enviando haces de luz a través de áreas públicas [CISCO SYSTEMS, INC, 2009].

- ✓ Área extensa (WAN): interconecta las LAN, que a su vez proporcionan acceso a los computadores o a los servidores de archivos ubicados en otros lugares. Como las WAN conectan redes de usuarios dentro de un área geográfica extensa, permiten que

las empresas se comuniquen entre sí a través de grandes distancias. Las WAN permiten que los computadores, impresoras y otros dispositivos de una LAN compartan y sean compartidas por redes en sitios distantes. Las WAN proporcionan comunicaciones instantáneas a través de zonas geográficas extensas. El software de colaboración brinda acceso a información en tiempo real y recursos que permiten realizar reuniones entre personas separadas por largas distancias, en lugar de hacerlas en persona. Las redes de área amplia también dieron lugar a una nueva clase de trabajadores, los empleados a distancia, que no tienen que salir de sus hogares para ir a trabajar [CISCO SYSTEMS, INC, 2009].

Las WAN están diseñadas para realizar lo siguiente:
- Operar entre áreas geográficas extensas y distantes
- Posibilitar capacidades de comunicación en tiempo real entre usuarios
- Brindar recursos remotos de tiempo completo, conectados a los servicios locales
- Brindar servicios de correo electrónico, World Wide Web, transferencia de archivos y comercio electrónico

Algunas de las tecnologías comunes de WAN son:
- Módems
- Red digital de servicios integrados (RDSI)
- Línea de suscripción digital (DSL - Digital Subscriber Line)
- Frame Relay
- Series de portadoras para EE.UU. (T) y Europa (E): T1, E1, T3, E3
- Red óptica síncrona (SONET)

9.2 LINUX

Linux es un sistema operativo compatible con Unix. Existen dos características que lo diferencian del resto de los sistemas que se encuentran en el mercado:

- ✓ Es gratuito, esto significa que no hay que pagar ningún tipo de licencia a ninguna casa desarrolladora de software por el uso del mismo.
- ✓ El sistema viene acompañado del código fuente. Además lo conforman el núcleo del sistema y un gran número de programas y librerías que hacen posible su utilización.

Linux se distribuye bajo la Licencia Pública General (General Public License, GNU), la cual puede leerse en http://www.gnu.org/copyleft/gpl.html. Esta licencia establece que el código fuente siempre debe estar disponible [MARTÍNEZ, 2001].

9.2.1 CARACTERÍSTICAS

Linux es un sistema operativo completo con tres características principales: estabilidad, confiabilidad y potencia, además de algunas otras características que lo hacen uno de los sistemas más poderosos del mercado [MARTÍNEZ, 2001].

- ✓ En Linux pueden correr varios procesos a la vez de forma ininterrumpida.
- ✓ Tiene un entorno de programación completo, incluyendo compiladores de C, C++, Pascal, FORTRAN, utilidades como Qt y lenguajes de scripting (guiones) como Perl, gwak y sed.
- ✓ Las distribuciones de Linux no son costosas, éstas pueden descargarse gratuitamente de los lugares apropiados en Internet, o bien comprar los CDs por muy poco dinero, comparado con otros sistemas comerciales.
- ✓ Es un sistema operativo multitarea, es decir, tiene la capacidad de ejecutar varios programas al mismo tiempo.
- ✓ Linux es multiusuario, esto es, puede mantener a varios usuarios que estén usando la misma máquina simultáneamente.
- ✓ Posee buen soporte de red.
- ✓ Normalmente permite trabajar con múltiples procesadores.

9.2.2 ADQUISICIÓN DEL SISTEMA OPERATIVO LINUX

Linux es un sistema operativo de distribución gratuita, por lo que los archivos y programas necesarios para su funcionamiento se pueden encontrar en multitud de servidores conectados a Internet. Debido a que reunir los archivos necesarios e instalarlos en el sistema llega a ser una tarea bastante complicada, existen las llamadas distribuciones de Linux, las cuales no son otra cosa que una recopilación de programas y archivos, organizados y preparados para su instalación.

Las diferencias entre una distribución y otra radican en la calidad de los paquetes incluidos en la misma, en que algunos programas de instalación pueden ser más fáciles de utilizar que otros, en la calidad de la documentación, en que el conjunto de programas y librerías sean compatibles entre sí, etc. Es interesante destacar que el kernel o núcleo es el mismo para todas las distribuciones, exceptuando la versión que se emplee, la cual puede variar [MARTÍNEZ, 2001].

Existen muchas distribuciones de Linux, pero las más comunes son:

- ✓ Red Hat Linux: Esta es una distribución que tiene muy buena calidad, contenidos y soporte a los usuarios por parte de la empresa que la distribuye. Es necesario el pago de una licencia de soporte. Es además enfocada a empresas [LINUX, 2006].

Red Hat Linux es instalado con un ambiente gráfico llamado Anaconda, diseñado para su fácil uso por novatos. También incorpora una herramienta llamada Lokkit para configurar las capacidades de Cortafuegos.
Red Hat Linux carece de muchas características debido a posibles problemas de derechos de autor y patentes. El soporte al formato NTFS está ausente, aunque puede ser instalado libremente [REDHAT, 2009].

- ✓ Debian Linux Distribution: Otra distribución con muy buena calidad. El proceso de instalación es quizás un poco mas complicado, pero sin mayores problemas. Gran estabilidad antes que últimos avances [LINUX, 2006].
  Según [DEBIAN, 2007], Debian se caracteriza por:
  - La disponibilidad en varias plataformas hardware.
  - Una amplia colección de software disponible.
  - Un grupo de herramientas para facilitar el proceso de instalación y actualización del software.
  - No tiene marcado ningún entorno gráfico en especial, ya sea GNOME, KDE u otro.

- ✓ S.u.S.E. Linux: Entre las principales virtudes de esta distribución se encuentra el que sea una de las más sencillas de instalar y administrar, ya que cuenta con varios asistentes gráficos para completar diversas tareas [SUSE, 2009].

- ✓ Fedora Core: también conocida como Fedora Linux, es una distribución Linux desarrollada por la comunidad Fedora y promovida por la compañía estadounidense Red Hat. El objetivo del proyecto Fedora es conseguir un sistema operativo de propósito general y basado exclusivamente en software libre con el apoyo de la comunidad Linux [LINUXCENTRO, 2007].

- ✓ Knoppix: es una distribución de Linux basada en Debian y utilizando KDE. A diferencia de la mayoría de las distribuciones Linux, Knoppix es un LiveCD y no requiere una instalación en el disco duro; el sistema puede iniciarse desde un simple CD [LINUXCENTRO, 2007]. Además, reconoce automáticamente la mayoría del hardware del ordenador soportado por Linux cuando se inicia. Se caracteriza por ser totalmente libre y con programas libremente. Se pueden almacenar hasta 2 gigabytes en el CD de forma comprimida, la descompresión es transparente. Knoppix utiliza el módulo cloop para funcionar a partir de una imagen comprimida, grabada en el CD-ROM.
  Knoppix también puede ser instalado en el disco duro utilizando un script de instalación. No obstante, ya que esto va más allá del propósito original de Knoppix, la instalación en el disco duro se recomienda sólo para usuarios avanzados. Existe también una distribución basada en Knoppix, llamada clusterKnoppix, que se utiliza para probar configuraciones en cluster [KNOPPIX, 2009].

- ✓ Ubuntu: es una distribucion basada en Debian, con lo que esto conlleva y centrada en el usuario final y facilidad de uso. Muy popular y con mucho soporte en la comunidad. El entorno de escritorio por defecto es GNOME [LINUX, 2006].

9.2.3 KERNEL

Según [MARTÍNEZ, 2001], el kernel o núcleo de Linux se puede definir como el corazón del sistema operativo. Es el encargado de que el software y el hardware del computador puedan trabajar juntos.

El kernel se encarga de administrar la memoria para todos los programas en ejecución, de la administración del tiempo de procesador que dichos programas en ejecución utilizan y es el encargado de administrar los accesos de entrada/salida.

En general existen dos versiones de kernel de Linux:

- ✓ Versión de producción: Es la versión estable hasta el momento. Esta versión es el resultado final de las versiones de desarrollo o experimentales.
  Cuando el equipo de desarrollo del núcleo experimental, decide que ha conseguido un núcleo estable y con la suficiente calidad, lanza una nueva versión de producción o estable y es la que se debería emplear para un uso normal del sistema.
- ✓ Versión de desarrollo: Esta versión es experimental y es la que utilizan los desarrolladores para programar, comprobar y verificar nuevas características, correcciones, etc. Estos núcleos suelen ser inestables y no deberían ser utilizados.

El kernel puede ser descargado de un gran número de servidores en Internet. El servidor principal es http://www.kernel.org

9.3 COMPUTACIÓN PARALELA

La creciente complejidad de los problemas tratados en ciertas áreas de la ciencia, la ingeniería, el procesamiento de datos, etc., requieren una gran capacidad de cálculo. Es un hecho bien establecido que para algunos problemas tales como los métodos numéricos, la simulación, la optimización, etc., la computación paralela ofrece soluciones eficientes. Las aplicaciones de esta área de la computación se extienden a diferentes disciplinas [DORMIDO, 2003].

Por lo general, el desarrollo de aplicaciones se realiza a partir de algoritmos que se ejecutan secuencialmente en un procesador. El problema surge cuando dichas aplicaciones ocupan

demasiado tiempo de CPU. Es ahí donde nace la idea del procesamiento paralelo [MARTÍNEZ, 2001].

Según [DORMIDO, 2003], la computación paralela o procesamiento en paralelo consiste en acelerar la ejecución de un programa mediante su descomposición en fragmentos que pueden ejecutarse de forma simultánea, cada uno en su propia unidad de proceso. Surge así, de forma natural, la idea de la computación paralela, que genéricamente consiste en utilizar n computadores para multiplicar, idealmente por n la velocidad computacional obtenida de un único computador. Por supuesto, esta es una situación ideal que muy rara vez se consigue en la práctica. Normalmente, los problemas no pueden dividirse perfectamente en partes totalmente independientes y se necesita, por tanto, una interacción entre ellas que ocasiona una disminución de la velocidad computacional. En este sentido se habla de mayor o menor grado de paralelismo en la medida en que un algoritmo sea más o menos divisible en partes independientes con igual costo computacional. Entre las interacciones hay que considerar principalmente las dos siguientes:

a) La transferencia de datos.
b) La sincronización de los cálculos de los diferentes procesadores.

Sin embargo, dependiendo de la naturaleza del problema y de su grado de paralelismo, se pueden conseguir grandes mejoras en el rendimiento. Hay que tener en cuenta que la computación paralela siempre se podrá beneficiar de los progresos y avances que se consigan en los procesadores individuales. El objetivo final de disponer de computadores paralelos más rápidos y potentes, es resolver categorías de problemas disminuyendo significativamente el tiempo de su resolución, hecho que no sería posible con los computadores serie actuales [DORMIDO, 2003].

Paralelizar un proceso consiste en mejorar el tiempo de ejecución de un programa específico mediante la minimización de un problema en problemas más simples [PÉREZ, 2001]. Esto implica tomar una gran tarea, dividirla en tareas menores y luego trabajar sobre cada una de ellas simultáneamente. El objetivo es completar la tarea principal en menos tiempo del que tomaría si se trabajara de una forma convencional. Un programa ejecutado en n procesadores llega a ser más rápido que en un solo procesador [MARTÍNEZ, 2001].

Mientras que un sistema de n procesadores paralelos es menos eficiente que un procesador n veces más rápido, el sistema paralelo a menudo es más barato de construir. El procesamiento paralelo es una solución excelente para tareas que requieren grandes cantidades de procesamiento, tienen restricciones de tiempo para finalizarse y especialmente para aquellas que pueden ser divididas en n hilos de ejecución. De hecho, en años recientes, la mayoría de los sistemas de procesamiento de alto rendimiento, también conocidos como supercomputadores, tienen arquitecturas paralelas [GOTTLIEB, 1989].

Si bien el procesamiento paralelo ofrece una ventaja definitiva en cuanto a costos, su principal beneficio, la escalabilidad, es decir, su capacidad de crecimiento, puede ser difícil

de alcanzar. Esto se debe a que conforme se añaden procesadores, las disputas por los recursos compartidos se intensifican [ROMO, 2003].

De [DORMIDO, 2003], el concepto de computador paralelo, como es de esperar, no es una idea nueva. Por ejemplo, Gil1 ya escribió acerca de la programación paralela en 1958, y un año después Holland planteó la posibilidad de que un computador pudiera ejecutar un número arbitrario de subprogramas simultáneamente. En 1963 Conway describe el diseño de un computador paralelo y su programación, y 40 años más tarde se siguen encontrando numerosísimos artículos con títulos y planteamientos similares a los que ya apuntaban estos precursores del campo de la computación paralela.

Pero hasta 1981 no se presentó el primer sistema paralelo comercial de la que se tienen noticias. Fue comercializada por BBN Computers Advanced (Bolt, Beranek and Newman Computers Advanced) y se llamó Butter-y. Era capaz de distribuir su trabajo entre 256 procesadores, en concreto eran microprocesadores Motorola 68000, que se conectaban a través de una red multietapa con memorias de 500 Kbytes por procesador. Se llegaron a vender alrededor de 35 máquinas, casi todas ellas a universidades y centros de investigación.

En los últimos años el rendimiento de los computadores paralelos ha aumentado significativamente. En la actualidad la tendencia general es que se construyan a partir de elementos diseñados para sistemas monoprocesador, con lo que los sistemas paralelos se benefician de los avances obtenidos para los serie. Como dato significativo, el rendimiento máximo de un computador paralelo en 1999 era de más de 2 Teraflops, unas 35 veces superior al de cinco años y unos meses antes [DORMIDO, 2003].

Teniendo en cuenta la definición básica de computación paralela, se hace necesario entender que la construcción de un sistema de procesamiento paralelo puede realizarse de varias formas diferentes, tanto al considerar los componentes que lo integran como al analizar las estrategias y los procedimientos para comunicar los diversos elementos. La complejidad de las diversas posibilidades hace que se presenten diferentes taxonomías basadas en diversos puntos de vista.

9.3.1 ARQUITECTURAS PARA LA COMPUTACIÓN PARALELA

Según [ROMO, 2003], el procesamiento paralelo ofrece una gran ventaja en cuanto a costos. Sin embargo, su principal beneficio, la escalabilidad (crecer hacia arquitecturas de mayor capacidad), puede ser difícil de alcanzar aún. Esto se debe a que conforme se añaden procesadores, las disputas por los recursos compartidos se intensifican.

Existen tres tipos de arquitecturas que soportan el procesamiento paralelo, cada una con sus propias ventajas y desventajas.

✓ Multiprocesamiento simétrico (SMP)

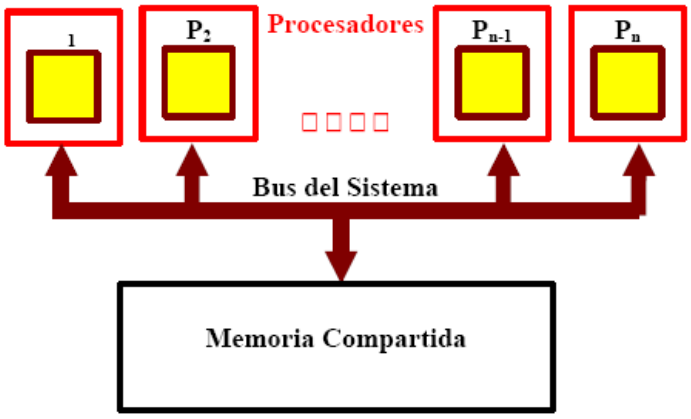

Figura 28. Multiprocesamiento Simétrico
Tomada de [ROMO, 2003]

El Multiprocesamiento Simétrico (Symmetric Multiprocessing, SMP) tiene un diseño simple, efectivo y económico (figura 28). En SMP, muchos procesadores comparten la misma memoria RAM y el bus del sistema. La presencia de un solo espacio de memoria simplifica tanto el diseño del sistema físico (hardware) como la programación de las aplicaciones (software). Esa memoria compartida permite que un Sistema Operativo con Multiconexión distribuya las tareas entre varios procesadores, o que una aplicación obtenga toda la memoria que necesita para una simulación compleja. La memoria globalmente compartida también vuelve fácil la sincronización de los datos.
SMP es uno de los diseños de procesamiento paralelo más maduros. Sin embargo, la memoria global contribuye al problema más grande de SMP: conforme se añaden procesadores, el tráfico en el bus de memoria se satura. Al añadir memoria caché a cada procesador se puede reducir algo del tráfico en el bus.
Al manejarse ocho o más procesadores, el cuello de botella se vuelve crítico, inclusive para los mejores diseños, por lo que SMP es considerada una tecnología poco escalable.

✓ Procesamiento masivamente paralelo (MPP)

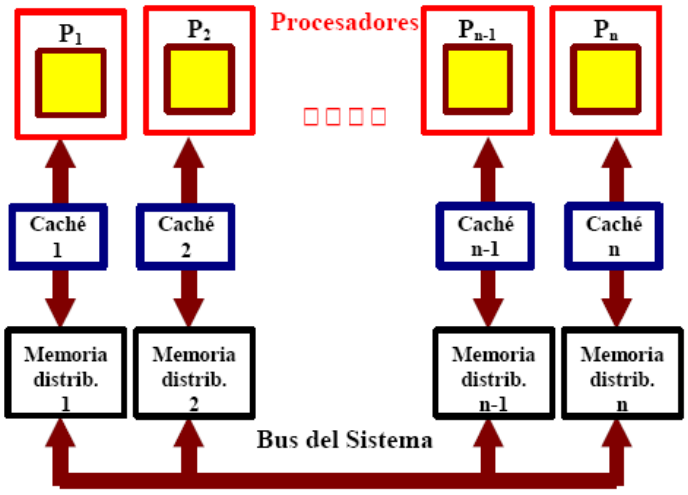

Figura 29. Procesamiento Masivamente Paralelo
Tomada de [ROMO, 2003]

El Procesamiento Masivamente Paralelo (Massively Parallel Processing, MPP) es una arquitectura computacional de alto rendimiento. Para evitar los cuellos de botella en el bus de memoria, MPP no utiliza memoria compartida; en su lugar, distribuye equitativamente la memoria RAM entre los procesadores de modo que se asemeja a una red: cada procesador con su memoria distribuida asociada es similar a un computador dentro de una red de procesamiento distribuido (figura 29).

Para tener acceso a las áreas de memoria fuera de su propia RAM, es decir, a memoria libre no empleada por los otros procesadores, los procesadores utilizan un esquema de paso de mensajes análogo a los paquetes de datos en redes. Este sistema reduce el tráfico del bus, debido a que cada sección de memoria interactúa únicamente con aquellos accesos que le están destinados, en lugar de interactuar con todos los accesos a memoria, como ocurre en un sistema SMP. Esto permite la construcción de sistemas MPP de gran tamaño, con cientos y aún miles de procesadores, por lo que MPP es una tecnología altamente escalable.

La parte negativa de MPP, desde el punto de vista tecnológico, es que la programación se vuelve difícil, debido a que la memoria se rompe en pequeños espacios separados. Sin la existencia de un espacio de memoria globalmente compartido, ejecutar una aplicación que requiere una gran cantidad de RAM, comparada con la memoria local, puede ser difícil. La sincronización de datos entre tareas ampliamente distribuidas también se complica, particularmente si un mensaje debe pasar por muchos componentes de hardware hasta alcanzar la memoria del procesador destino.

El costo de las soluciones basadas en MPP es mucho más alto que el costo por procesador de las soluciones SMP, por lo que su uso sólo se justifica cuando la necesidad de procesamiento es muy alta.

✓ Procesamiento paralelo escalable (SPP)

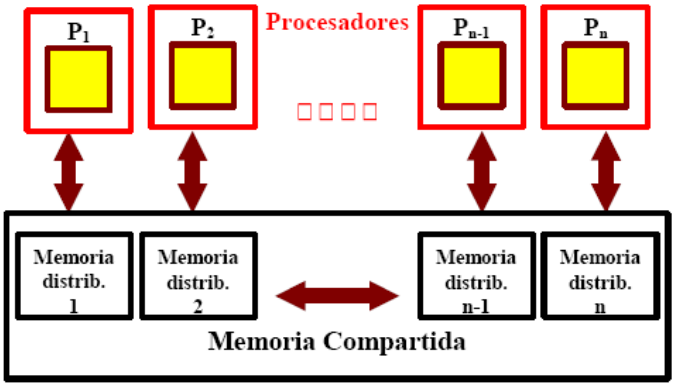

Figura 30. Procesamiento Paralelo Escalable
Tomada de [ROMO, 2003]

La tercera arquitectura paralela, el Procesamiento Paralelo Escalable (Scalable Parallel Processing, SPP), es un híbrido de SMP y MPP, que utiliza una memoria jerárquica de dos niveles para alcanzar la escalabilidad. La primera capa consiste de componentes de memoria distribuida que son esencialmente parte de sistemas MPP completos, con múltiples nodos, entendiendo por nodo la unión de un procesador y su memoria distribuida correspondiente. El segundo nivel de memoria está globalmente compartido al estilo SMP.

Se construyen sistemas SPP grandes interconectando dos o más nodos a través de la segunda capa de memoria, de modo que esta capa aparece, lógicamente, como una extensión de la memoria individual de cada nodo (figura 30).
La memoria de dos niveles reduce el tráfico de bus debido a que solamente ocurren actualizaciones para mantener coherencia de memoria. Por lo tanto, SPP ofrece la facilidad de programación del modelo SMP, a la vez que provee una escalabilidad similar a la de un diseño MPP.

9.3.2 TAXONOMÍA DE ARQUITECTURAS PARALELAS

De [PROYECTO DE GRADO, 2006], las arquitecturas paralelas superaran las limitaciones físicas sobre velocidades de reloj y cantidad de memoria de un sistema secuencial, permitiendo la resolución de problemas cada vez más grandes y complejos. Estas arquitecturas permiten disminuir los tiempos que un componente del sistema está en desuso.

Para aprovechar al máximo el poder de procesamiento de un computador con arquitectura paralela es necesario utilizar el paradigma de programación paralela, debido que permite definir procesos que el computador ejecutará en forma concurrente. El paradigma de programación paralela permite el aprovechamiento de problemas que son concurrentes en

forma natural y que se pueden dividir en varios eventos más pequeños ocurriendo en paralelo, como el estudio de órbitas planetarias o la simulación del clima.

Las arquitecturas paralelas son alimentadas por diferentes avances tecnológicos como el incremento de poder de procesamiento (por ejemplo la creación de equipos multiprocesadores). La creación de redes con menores latencias (tiempo necesario para que un paquete de información viaje desde la fuente hasta un destino) y mayores anchos de banda (capacidad en la transmisión de datos) estimulan la creación de computadores paralelos más rápidos y baratos. La implementación de bibliotecas estimula el desarrollo del modelo de programación paralela.

Según [DORMIDO, 2003], existen diferentes criterios de clasificación, los cuales diferencian unos sistemas de otros por ejemplo, la taxonomía de Flynn y la red de interconexión.

9.3.3 ARQUITECTURA DE LOS COMPUTADORES SECUENCIALES

9.3.3.1 TAXONOMÍA DE FLYNN

En 1966 Flynn propone una clasificación generalista de los computadores, la cual adopta como criterio el flujo de instrucciones y el flujo de datos que en ellos se desarrolla. Se entiende por flujo (stream) una secuencia de elementos, en este caso de datos o de instrucciones.
Esta clasificación se denomina Taxonomía de Flynn y se puede resumir en la figura siguiente (figura 31):

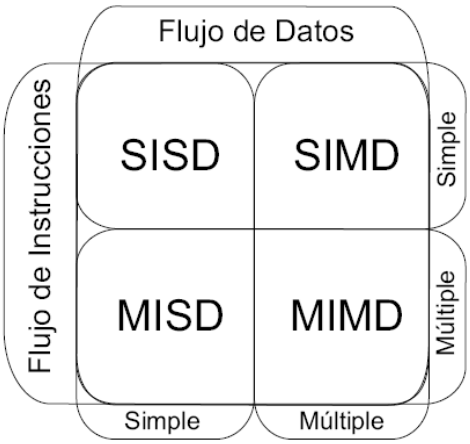

Figura 31. Taxonomía de Flynn
Tomada de [AKL, 1989]

La clasificación clásica propuesta por Flynn se basa en el flujo de instrucciones y en el flujo de datos, es decir, el mecanismo de control utilizado. Sin embargo, existe una gran cantidad de criterios, aunque no todos mutuamente excluyentes entre sí, para establecer una clasificación de las distintas arquitecturas paralelas. Así, tomando como base la clasificación de Flynn se atiende a la organización del espacio de memoria, analizando las diferentes posibilidades que permiten establecer diversas arquitecturas [DORMIDO, 2003].

- ✓ SISD (Single Instruction, Single Data), instrucción única, datos únicos. En este grupo se encuentran la mayoría de los computadores serie disponibles actualmente. Las instrucciones se ejecutan secuencialmente pero pueden estar solapadas en las etapas de ejecución (segmentación encauzada). Los computadores SISD pueden tener más de una unidad de recursos de cálculo, pero todas ellas están bajo el control de una única unidad de control [DORMIDO, 2003].
  Un computador de esta clase consiste básicamente de una unidad de procesamiento simple recibiendo un flujo de instrucciones simple que opera en un flujo de datos simple (figura 32). Generalmente se les puede ver como el conjunto de computadores tradicionales, que satisfacen la arquitectura Von Neumann, ya que fueron inventados por él y sus colaboradores a finales de los años cuarenta. Cada paso durante la unidad de control de computación emite una instrucción que opera en un dato obtenido desde la unidad de memoria. Un algoritmo para un computador de esta clase es llamado secuencial o serial [AKL, 1989].

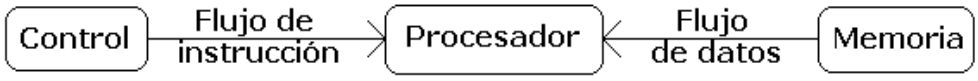

Figura 32. Computadores SISD
Tomada de [AKL, 1989]

- ✓ SIMD (Single Instruction, Multiple Data), instrucción única, datos múltiples. En esta clase se sitúan los procesadores matriciales en los que existen varias unidades de procesamiento trabajando sobre flujos de datos distintos pero ejecutando la misma instrucción proporcionada por una única unidad de control. Estos computadores garantizan sincronismo entre procesadores después de cada ciclo de ejecución de las instrucciones [DORMIDO, 2003]. Consisten de N procesadores idénticos y una única Unidad de Control. Cada uno de esos N procesadores posee su propia memoria local en donde puede almacenar programas y datos (figura 33).
  Este tipo de computador posee una unidad que controla el flujo de instrucciones que se ejecutan en diferentes unidades de procesamiento, bajo un único ciclo de reloj. Como cada unidad de procesamiento opera con sus propios datos, se nota la existencia de varios flujos de datos.
  Todos los procesadores operan dentro del control de flujo de instrucción simple distribuido o publicado por una unidad de control. Equivalentemente los N

procesadores pueden ser asumidos para tomar copias idénticas de un programa simple, cada copia del procesador esta almacenada en su memoria local.
La misma instrucción se ejecuta sincrónicamente por todas las unidades de procesamiento. Requiere menos hardware porque sólo necesita una unidad de control global y menos memoria porque tiene una sola copia del programa [AKL, 1989].

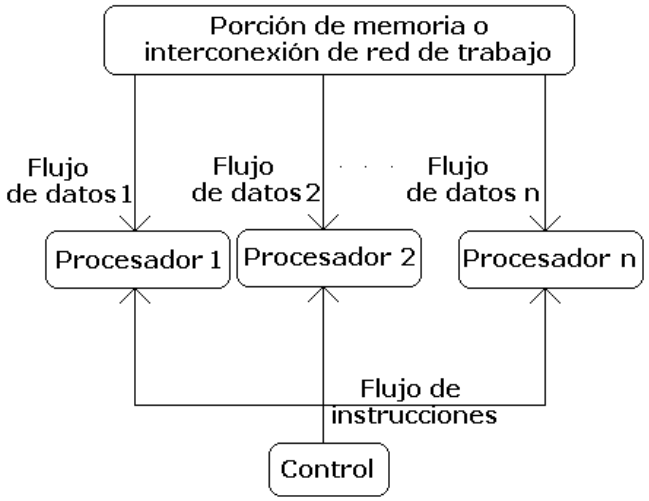

Figura 33. Computadores SIMD
Tomada de [AKL, 1989]

- ✓ MISD (Multiple Instruction, Single Data), instrucciones múltiples, datos únicos. Se caracteriza por la existencia de varias unidades de procesamiento cada una ejecutando una instrucción diferente pero sobre el mismo flujo de datos. En sentido estricto no se conoce ninguna materialización real de esta categoría, ya que limita las posibilidades del hardware sin claros beneficios sobre otros modelos. Se pueden considerar como un método de diseño de computadores de propósito especial para equilibrar recursos, ancho de banda de E/S y cálculo. Basándose en la segmentación, los datos fluyen en etapas desde memoria a un arreglo de unidades de cálculo y vuelven a memoria [DORMIDO, 2003].
  Tal y como se aprecia en la figura 34, un computador de esta clase consiste básicamente en N procesadores con su propia porción de unidad de control y con una unidad de memoria común (N flujos de instrucciones y un flujo de datos).
  Un dato recibido desde memoria es operado por todos los procesadores simultáneamente, cada uno de acuerdo a la instrucción que recibe desde su control.
  Este paralelismo es obtenido para permitir a los procesadores hacer diferentes cosas al mismo tiempo y con datos similares.
  Esta clase de computadores requiere una entrada que es dominada por varias operaciones [AKL, 1989].

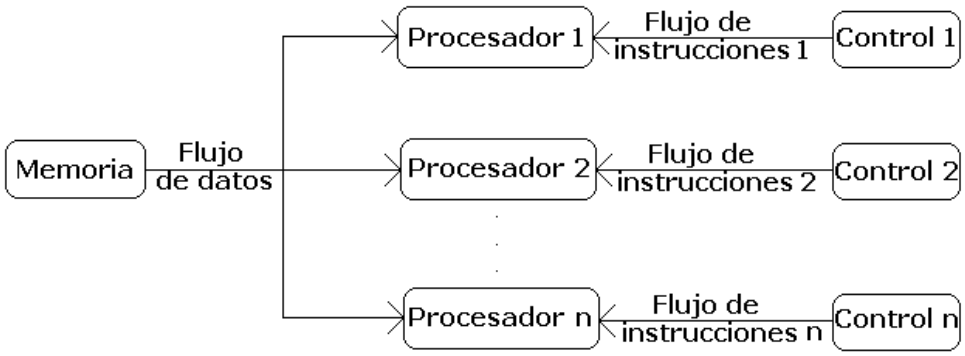

Figura 34. Computadores MISD
Tomada de [AKL, 1989]

- ✓ MIMD (Multiple Instruction, Multiple Data), instrucciones múltiples, datos múltiples. En esta categoría se incluyen la mayoría de los sistemas multiprocesadores y multicomputadores, y en ellos cada procesador es capaz de ejecutar un programa independiente de los demás procesadores. Un computador MIMD intrínsecamente implica interacciones entre varios procesadores porque todos los flujos de memoria se obtienen de un espacio de datos compartido por todos ellos [DORMIDO, 2003].
  Esta clase de computadores es la más general y la más poderosa en el paradigma de computación paralela que clasifica computadores paralelos de acuerdo a sí la instrucción y/o el flujo de datos es duplicado.
  La figura 35 muestra que se tienen N procesadores, N flujos de instrucciones y N flujos de datos [AKL, 1989].

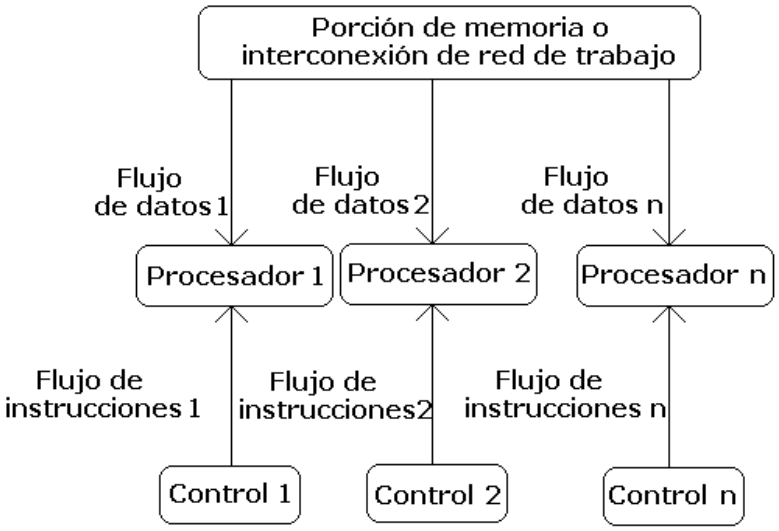

Figura 35. Computadores MIMD
Tomada de [AKL, 1989]

De acuerdo a [DORMIDO, 2003], se concluye que atendiendo a esta clasificación las arquitecturas paralelas pueden ser esencialmente de dos tipos: SIMD y MIMD. Se analizaran los aspectos generales de cada una de ellas.

a) Modelo SIMD

En el modelo SIMD cada procesador ejecuta en todo momento la misma operación, pero únicamente sobre sus propios datos. Las instrucciones del programa se envían a más de un procesador y cada uno actúa básicamente como un procesador aritmético-lógico sin una unidad de control. Una unidad de control independiente, separada, es la responsable de buscar las instrucciones en memoria y de repartirlas a los distintos procesadores. Cada procesador ejecuta la misma instrucción en sincronismo pero utilizando diferentes datos.

En un modelo SIMD, el conjunto de datos forma un arreglo y una instrucción actúa sobre todo el arreglo en un ciclo de instrucción. De ahí que el factor determinante en el desarrollo de este modelo sea el gran número de aplicaciones que trabajan sobre arreglos de datos. Por ejemplo, muchas de las simulaciones de sistemas físicos (sistemas moleculares, predicción del tiempo, etc.) operan sobre grandes arreglos de datos.

Las máquinas SIMD son apropiadas para el tratamiento de arreglos en bucles, pero su mayor debilidad se encuentra en el tratamiento de las sentencias de selección, en las que cada unidad de ejecución debe realizar una operación diferente sobre su dato, dependiendo del dato que se tenga. Las unidades de ejecución con el dato erróneo son inhabilitadas para que las unidades adecuadas puedan continuar. Estas situaciones se ejecutan a l/n del rendimiento, donde n es el número de selecciones a realizar. La poca flexibilidad del modelo SIMD ha provocado que en los últimos años se encuentre relegado a entornos muy concretos y especializados.

b) Modelo MIMD

En el modelo MIMD cada procesador actúa de forma esencialmente independiente. Uno o varios procesadores pueden, por ejemplo, estar transformando datos mediante un algoritmo complejo, mientras otro está generando una salida gráfica de resultados. Esto es lo que se denomina descomposición de control, de manera que cada procesador puede estar ejecutando un programa distinto. Mucho más habitual es la descomposición de datos, en la que los datos del problema se reparten entre los distintos procesadores y todos ejecutan el mismo programa sobre la zona de datos que se le ha asociado. Esta particularización del modelo MIMD se conoce como SPMD (Single Program, Multiple Data), programa único sobre múltiples datos. Es importante observar la diferencia con el modelo SIMD. En un sistema SPMD todos los procesadores ejecutan el mismo programa, pero esto no quiere decir que ejecute el mismo código simultáneamente. De este modo, un procesador puede estar ejecutando código distinto que el ejecutado por otro procesador, debido por ejemplo a una bifurcación condicional dependiente del resultado de una operación. Así, suponiendo

que dos procesadores llegan a una sentencia IF simultáneamente, para uno de ellos puede resultar un valor TRUE en la condición mientras que para el otro puede ser FALSE. En ese momento cada procesador sigue ejecutando un código diferente del mismo programa. Sin embargo, en un sistema SIMD el código de la sentencia IF ... THEN ELSE debe ejecutarse secuencialmente. Este modelo es, por tanto, mucho más flexible que el SIMD.

Una analogía que sirve para aclarar el funcionamiento de un modelo MIMD es considerarlo equivalente a la división de un proyecto en grupos de trabajo. Cada grupo se ocupa únicamente de su tarea, pero todos los grupos tienen que reunirse de vez en cuando para intercambiar conclusiones y coordinar el trabajo posterior. El intercambio de información entre procesadores depende del sistema de que se disponga.

c) Comparación de los Sistemas SIMD y MIMD

Evidentemente los sistemas SIMD necesitan menos hardware que los MIMD, ya que sólo requieren una unidad de control, y además necesitan menos memoria ya que sólo es necesario almacenar una copia del programa en ejecución. Los sistemas MIMD almacenan una copia del programa en cada procesador, y con ello el software asociado al programa, como por ejemplo el sistema operativo.

Por otro lado, los sistemas SIMD necesitan menos tiempo de inicio para las comunicaciones entre procesadores, debido a que son transferencias entre registros del procesador origen y destino.

Evidentemente, la desventaja de los sistemas SIMD es que no pueden ejecutar diferentes instrucciones en el mismo ciclo de reloj. Además, aunque en un principio puede parecer que los sistemas MIMD tienen un costo muy superior a los SIMD, esto en general no es cierto. Debido a que los sistemas MIMD pueden construirse con procesadores de propósito general, pueden beneficiarse de las ventajas económicas de una distribución para el mercado y el consumo en general, mientras que las arquitecturas SIMD pueden considerarse en este sentido específicas, con lo que su costo de producción suele ser mayor de lo que cabría esperar. De hecho, la unidad de control de un sistema SIMD debe ser diseñada específicamente. Con todo ello y debido a la economía de escala, las arquitecturas MIMD suelen tener un costo menor que un sistema SIMD y ofrecen mayores rendimientos y posibilidades. Incluso para las aplicaciones en las que se requiere sincronismo entre los procesadores, característica típica de los sistemas SIMD, se han desarrollado arquitecturas MIMD que incorporan la posibilidad de trabajar en modo SIMD mediante la inclusión del hardware adecuado.

Atendiendo a todas estas consideraciones puede observarse que los sistemas MIMD son más generales y eficaces.

9.3.3.2 ORGANIZACIÓN DEL ESPACIO DE DIRECCIONES DE MEMORIA

El intercambio de información entre procesadores depende del sistema de almacenamiento que se disponga. Atendiendo a este criterio se obtiene una nueva clasificación de las arquitecturas paralelas en:

- Sistemas de memoria compartida o multiprocesadores.
- Sistemas de memoria distribuida o multicomputadores.

✓ Sistemas de memoria compartida o multiprocesadores: De [DORMIDO, 2003], los procesadores de los sistemas con memoria compartida se caracterizan por compartir físicamente la memoria, es decir, todos acceden al mismo espacio de direcciones. Un valor escrito en memoria por un procesador puede ser leído directamente por cualquier otro. En principio, en esta arquitectura la memoria es igualmente accesible por todos los procesadores a través de la red de interconexión.

En este contexto, en el que la red de interconexión es determinante para la eficacia del sistema, son fundamentales dos parámetros que caracterizan los sistemas paralelos: la latencia de red y el ancho de banda. Ambos determinan la velocidad en la transferencia de datos entre elementos básicos del sistema paralelo. El primero se define como el tiempo que se tarda en enviar un mensaje a través de la red de interconexión del sistema paralelo, mientras que, en este contexto, se define el ancho de banda como el número de bits que se pueden enviar por unidad de tiempo.

Para garantizar la eficacia de esta arquitectura es fundamental que el ancho de banda sea elevado, ya que en cada ciclo de instrucción cada procesador puede necesitar acceder a la memoria a través de la red de interconexión. Este acceso a memoria puede ser lento debido a que las solicitudes de lectura o de escritura pueden tener que pasar por varias etapas en la red. En la arquitectura de memoria compartida se satisface que la latencia de red es baja, y el ancho de banda alto, sólo si no se encuentran varios procesadores tratando de acceder al medio utilizado para la transmisión de datos simultáneamente. Por este hecho y otros problemas relacionados con el mismo, el número de procesadores en este tipo de sistemas suele ser pequeño y la arquitectura más común para la comunicación es la de bus, en la cual todos los procesadores y módulos de memoria se conectan a un único bus. En esta arquitectura el tiempo de acceso a memoria es el mismo para cualquier palabra, por lo que se denomina arquitectura de acceso uniforme, UMA (Uniform Memory Access).

La mayoría de los sistemas de memoria compartida incorporan una memoria cache local en cada procesador, y del mismo modo que en los computadores secuenciales, sirve para aumentar el ancho de banda entre el procesador y la memoria local. Análogamente puede existir una memoria cache para la memoria global. Cuando se utilizan memorias cache es fundamental asegurar la coherencia de la información en la memoria cache, de modo que cuando un procesador modifique el valor de una variable compartida los demás procesadores no consideren su valor anterior, ya que es incorrecto desde ese momento. Sin embargo, existe otro factor que limita a la

arquitectura UMA: la escalabilidad. La escalabilidad es un factor fundamental de los sistemas paralelos. De forma genérica puede entenderse como la capacidad del sistema para mejorar la potencia de cálculo cuando el número de nodos aumenta. En el caso ideal sería lineal, Sin embargo, debido a factores como los tiempos de comunicación, la escalabilidad no suele ser lineal y llega a saturarse (esto significa, que cuando se añaden más nodos al sistema, no se consigue ninguna mejora).

Este tipo de memoria compartida no es adecuada para sistemas escalables debido a los problemas de acceso a la memoria remota que presentan. Entre los sistemas implementados destaca el Encore Multimax de Encore Computer Corporation que representa la tecnología de los años 80 y el Power Challenge que es representativo de los 90.

En general, los sistemas UMA presentan una escalabilidad limitada incluso cuando incorporan mecanismos de memoria cache.

Sin embargo, algunas mejoras pueden realizarse dotando a cada procesador de una memoria local, en la que se almacena el código que se está ejecutando en el procesador y aquellos datos que no tengan que ser compartidos por los otros procesadores y, por tanto, son locales a ese procesador. Mediante esta estrategia se evitan accesos a memoria a través de la red de interconexión relacionados con búsquedas de código y datos locales, lo que mejora la eficacia del sistema. Esta modificación de la arquitectura es conocida como memoria compartida con acceso no uniforme a memoria. Teniendo en cuenta este esquema y considerando el tiempo que un procesador necesita para acceder a memoria local y global, las memorias compartidas se clasifican en memorias de acceso uniforme, UMA, y no uniforme, NUMA.

Los sistemas de memoria compartida de acceso no uniforme son también conocidos como sistemas de memoria compartida distribuida, DSM (Distributed Shared Memory). La idea de distribuir la memoria compartida puede llevarse al límite eliminando completamente el bloque de memoria compartida común (global), siempre y cuando los tiempos de acceso a las memorias locales sean muy inferiores al tiempo de acceso por la red de interconexión. En este diseño los accesos de un procesador a las memorias locales de los otros se realizan mediante el control de un hardware específico. Obviamente esta arquitectura es NUMA.

En general, el esquema de la arquitectura NUMA no requiere del empleo de mecanismos especiales que aseguren que todos los nodos tienen un conocimiento coherente del espacio de direcciones global. Es posible manejar un espacio común de direcciones y que sea el programador el encargado de manipular el contenido de esas direcciones de memoria evitando cualquier falta de coherencia. Evidentemente, en las arquitecturas NUMA la red de interconexión juega un papel fundamental. Es necesario señalar que, dado que cada procesador ve un espacio común de direcciones a nivel lógico pero que realmente está distribuido a nivel físico, cada uno de los accesos a posiciones de memoria que se encuentran fuera del rango mantenido por el nodo local se convertirá en accesos remotos. La comunicación entre nodos se realiza a través de peticiones de bloques de forma transparente al programador. Para que el sistema sea eficaz, la latencia de la red de interconexión no debe incrementar los tiempos de los accesos remotos excesivamente ya que, aunque el número de accesos remotos fuese

muy inferior al de accesos locales, un costo elevado en cada fallo afectaría de forma drástica al rendimiento global del sistema.

Evidentemente el concepto genérico de memoria compartida puede aplicarse tanto a sistemas SIMD como MIMD, aunque por las razones explicadas anteriormente el interés se centra en los MIMD. A la combinación de MIMD y memoria compartida se le denomina multiprocesadores simétricos (SMP, Symmetric Multiprocessors). Los sistemas SMP suelen tener de 2 a 64 procesadores, y se les suele conocer como la arquitectura que comparte todo, es decir, todos los procesadores utilizan conjuntamente la totalidad de los recursos que tenga disponible el sistema (bus, memoria, módulos de entrada y salida, etc.). Por otro lado, en los sistemas SIMD únicamente se ejecuta una copia del sistema operativo.

- ✓ Sistemas de memoria distribuida o multicomputadores: De [DORMIDO, 2003], en los sistemas de memoria distribuida cada procesador dispone de su propia memoria, denominada local o privada, independiente del resto y accesible sólo por su procesador. La comunicación se realiza por paso de mensajes, es decir, para que un dato que reside en la memoria de un procesador pase a la de otro, el primero debe construir un mensaje por software, enviarlo a través de una red de interconexión y el segundo debe recibirlo. Como puede observarse es un mecanismo más complejo que con memoria compartida. Esta arquitectura es también conocida como arquitectura de memoria privada o arquitectura de paso de mensajes.
  El concepto de paso de mensajes parece ser la estrategia dominante en los sistemas con gran número de procesadores (mayor que 100), y es especialmente útil en entornos donde la ejecución de los programas puede dividirse en pequeños subprogramas independientes. La latencia de red puede ser alta, pero para analizar el ancho de banda es necesario atender a otro factor de eficiencia de los sistemas paralelos conocido como granularidad del computador paralelo. En general, se entiende por granularidad del computador paralelo el cociente entre el tiempo requerido para realizar una operación básica de comunicación y el tiempo requerido para realizar una operación básica de cálculo de los procesos. Para un tiempo dado de operación básica de cálculo ofrece una medida del número y del tamaño de los paquetes de información utilizados en las comunicaciones. En los sistemas de paso de mensajes para lograr un uso adecuado del ancho de banda es necesario realizar un cuidadoso reparto de los datos sobre los procesadores con el fin de disminuir la granularidad de las comunicaciones. Por disminuir la granularidad de las comunicaciones se entiende minimizar el número de mensajes y maximizar su tamaño. Evidentemente, aunque el concepto genérico puede aplicarse a sistemas SIMD, también es aplicable a sistemas MIMD. A la combinación de sistema MIMD con arquitectura de paso de mensajes se le conoce como multicomputadores. Los sistemas con memoria distribuida o multicomputadores pueden, a su vez, ser un único computador con múltiples CPUs comunicadas por un bus de datos o bien múltiples computadores, cada uno con su propio procesador, enlazados por una red de interconexión más o menos rápida. En el primer caso se habla de procesadores masivamente paralelos (MPPs, Massively Parallel Processors),

y en el segundo se conocen de forma genérica como cluster. Un cluster, a nivel básico, es una colección de estaciones de trabajo O PCS interconectados mediante algún sistema de red de comunicaciones. En la literatura existente de arquitecturas paralelas se utilizan numerosos nombres para referirse a un cluster, entre los que destacan:

- Redes de estaciones de trabajo (NOWs, Network of Workstations)
- Redes de PCs (NOPCs, Network of PCs)
- Cluster de estaciones de trabajo (COWs, Cluster of Workstations)
- Cluster de PCs (COPCs, Cluster of PCs)

Realizando una clasificación estricta, los clusters pueden ser de dos tipos dependiendo de si cada computador del cluster está o no exclusivamente dedicado a él. Si es así, se habla de un cluster de clase Beowulf. En cualquier otro caso se suele definir al cluster como NOW. En muchas ocasiones los términos cluster y Beowulf se confunden y se utilizan indistintamente. El término Beowulf lo utilizó la NASA para dar nombre a un proyecto que pretendía construir un ordenador capaz de alcanzar el gigaflops a partir de componentes asequibles. Y lo consiguieron, el Beowulf original de 1994, con 16 procesadores 486 DX4 ejecutando Linux, fue capaz de alcanzar 1,25 Gigaflops. El término no se debe a razones técnicas. Beowulf era un guerrero escandinavo del siglo VI cuyas aventuras se relatan en el primer texto conocido en lengua inglesa (similar al Cantar del Mío Cid en la lengua española). Las características más relevantes de los sistemas Beowulf son las siguientes:

- Un sistema Beowulf es un conjunto de nodos minimalistas (los cuales constan solamente de una placa madre, una CPU, las memorias y algún dispositivo de comunicaciones) conectados por un medio de comunicación barato, en el que la topología de la red se ha diseñado para resolver un tipo de problema específico.
- Cada nodo de un Beowulf se dedica exclusivamente a procesos del supercomputador.
- En una red de estaciones de trabajo (NOWs) suele existir un switch central para realizar las comunicaciones, mientras que en un Beowulf el mecanismo es más rudimentario: conexiones placa a placa por cable RJ-45 cruzado.
- La programación de un Beowulf es fuertemente dependiente de la arquitectura y siempre se realiza por paso de mensajes.
- Para programar un Beowulf en primer lugar se diseña el modelo de paralelismo, se observan cómo son las comunicaciones entre los nodos y se implementan físicamente. De esta forma se evita el hardware innecesario a cambio de una fuerte dependencia entre el software y la topología de la red. En caso de que cambie el problema hay que recablear el sistema.

9.3.4 SISTEMAS PARALELOS

Basado en [DE GIUSTI, 2005], se percibe que en la actualidad es innegable la importancia y el creciente interés en el procesamiento paralelo y distribuido dentro de la Ciencia de la Computación por un gran número de razones: crecimiento de la potencia de cálculo dada

por la evolución tecnológica, transformación y creación de algoritmos que explotan la concurrencia para obtener mejores tiempos de respuesta, necesidad de tratar sistemas de tiempo real distribuidos, el límite físico de las máquinas secuenciales que convierte a la solución paralela en la única factible, entre otras.

En términos generales las máquinas paralelas permiten resolver problemas de complejidad creciente y obtener resultados con mayor velocidad. Estas máquinas ofrecen soluciones más rápidas, permitiendo resolver problemas más grandes y complejos cuyos datos de entrada o resultados intermedios exceden la capacidad de memoria de un procesador, las simulaciones pueden ser ejecutadas con mayor resolución, los fenómenos físicos pueden ser modelados de manera más realista. En algunos casos el costo del paralelismo puede ser alto en términos del esfuerzo de programación requerido, en muchos casos debe pensarse en la aplicación de técnicas novedosas reescribiendo completamente el código serial, y las técnicas de depuración y de ajuste en el desempeño "secuenciales" no se extienden fácilmente al mundo paralelo.

Se debe hacer referencia a sistemas paralelos como la combinación de algoritmo y arquitectura, ya que a diferencia del cómputo secuencial donde el modelo RAM es aceptado prácticamente como estándar, en el paralelismo no se encuentra un modelo unificador ya que cada uno enfatiza determinados aspectos en detrimento de otros. Las diferentes arquitecturas (memoria compartida o distribuida, procesadores homogéneos o heterogéneos, topologías de conexión, etc.) hacen que en el desarrollo de las soluciones paralelas sea indispensable considerar la combinación de ambos factores.

La última etapa en el desarrollo de algoritmos paralelos es el mapeo de procesos en procesadores; los objetivos son mejorar la utilización de los procesadores y obtener el mejor tiempo de respuesta de la aplicación realizando la distribución de manera que la carga computacional tienda a ser equitativa (balanceada) en el tiempo. Este es uno de los aspectos centrales del procesamiento paralelo, pues producen un impacto directo en el uso eficiente de los recursos (que implican costo) y en las mejoras de desempeño, con el fin de que sean alcanzables.

El desempeño obtenido en el sistema paralelo está dado por una compleja relación en la que intervienen gran cantidad de factores: tamaño del problema, arquitectura de soporte, distribución de procesos en procesadores, existencia o no de un algoritmo de balanceo de carga, etc. Existen numerosas métricas para evaluar sistemas paralelos; entre ellas se encuentran tiempo efectivo, speedup, eficiencia, producto procesador-tiempo (costo), overhead paralelo, grado de concurrencia, escalabilidad, etc.

9.3.4.1 FORMAS DE PARALELIZACIÓN

Según [HIDALGO, 2004], existen diferentes formas para paralelizar un programa:

- ✓ Paralelización de grano fino: La paralelización del programa se realiza a nivel de instrucción. Cada procesador hace una parte de cada paso del algoritmo.

- ✓ Paralelización de grano medio: Los programas se paralelizan a nivel de bucle. Esta paralelización se realiza habitualmente de una forma automática en los compiladores.

- ✓ Paralelización de grano grueso: Se basan en la descomposición del dominio de datos entre los procesadores, siendo cada uno de ellos el responsable de realizar los cálculos sobre sus datos locales.
  La paralelización de grano grueso tiene como atractivo la portabilidad, ya que se adapta perfectamente tanto a multiprocesadores de memoria distribuida como de memoria compartida.

  Este tipo de paralelización se puede a su vez realizar siguiendo tres estilos distintos de programación: paralelismo en datos, programación por paso de mensajes y programación por paso de datos.
  - Paralelismo en datos: El compilador se encarga de la distribución de los datos guiado por un conjunto de directivas que introduce el programador. Estas directivas hacen que cuando se compila el programa las funciones se distribuyan entre los procesadores disponibles. Como principal ventaja presenta su facilidad de programación. Los lenguajes de paralelismo de datos más utilizados son el estándar HPF (High Performance Fortran) y el OpenMP.
  - Programación por paso de mensajes: El método más utilizado para programar sistemas de memoria distribuida es el paso de mensajes o alguna variante del mismo. La forma más básica consiste en que los procesos coordinan sus actividades mediante el envío y la recepción de mensajes. Las librerías más utilizadas son por este orden la estándar MPI (Message Passing Interface) y PVM (Parallel Virtual Machine).
  - Programación por paso de datos: A diferencia del modelo de paso de mensajes, la transferencia de datos entre los procesadores se realiza con primitivas unilaterales tipo put-get, lo que evita la necesidad de sincronización entre los procesadores emisor y receptor. Es un modelo de programación de muy bajo nivel pero muy eficiente, aunque en la actualidad son muy pocos los fabricantes que los soportan.

9.3.5 PROGRAMAS PARALELOS

De [HIDALGO, 2004], un programa es paralelo si en cualquier momento de su ejecución puede ejecutar más de un proceso. Para crear programas paralelos eficientes hay que poder crear, destruir y especificar procesos así como la interacción entre ellos.

En las ciencias de la computación, un algoritmo paralelo, en oposición a los algoritmos clásicos o algoritmos secuenciales, es un algoritmo que puede ser ejecutado por partes en el

mismo instante de tiempo por varias unidades de procesamiento, para finalmente unir todas las partes y obtener el resultado correcto.

Según [BOONIC, 2008], los algoritmos paralelos son importantes porque es más rápido tratar grandes tareas de computación mediante la paralelización que mediante técnicas secuenciales. Esta es la forma en que se trabaja en el desarrollo de los procesadores modernos, ya que es más difícil incrementar la capacidad de procesamiento con un único procesador que aumentar su capacidad de cómputo mediante la inclusión de unidades en paralelo, logrando así la ejecución de varios flujos de instrucciones dentro del procesador. Pero hay que ser muy prudente con la excesiva paralelización de los algoritmos ya que cada algoritmo paralelo tiene una parte secuencial y debido a esto, los algoritmos paralelos pueden llegar a un punto de saturación. Por todo esto, a partir de cierto nivel de paralelismo, añadir más unidades de procesamiento puede sólo incrementar el costo y la disipación de calor.

El costo o complejidad de los algoritmos secuenciales se estima en términos del espacio (memoria) y tiempo (ciclos de procesador) que requiera. Los algoritmos paralelos también necesitan optimizar la comunicación entre diferentes unidades de procesamiento. Esto se consigue mediante la aplicación de dos paradigmas de programación y diseño de procesadores distintos: memoria compartida o paso de mensajes.

La técnica memoria compartida necesita del uso de cerrojos en los datos para impedir que se modifique simultáneamente por dos procesadores, por lo que se produce un costo extra en ciclos de CPU desperdiciados y ciclos de bus. También obliga a serializar alguna parte del algoritmo.

La técnica paso de mensajes usa canales y mensajes pero esta comunicación añade un costo al bus, memoria adicional para las colas y los mensajes y latencia en el mensaje. Los diseñadores de procesadores paralelos usan buses especiales para que el costo de la comunicación sea pequeño pero siendo el algoritmo paralelo el que decide el volumen del tráfico [BOONIC, 2008].

Una de las subclases con mayor predominancia en los algoritmos paralelos, es la de los algoritmos distribuidos, los cuales son algoritmos diseñados para trabajar en entornos tipo clusters y de computación distribuida, donde se usan otras técnicas, fuera del alcance de los algoritmos paralelos clásicos.

Según [PÉREZ, 2003], el paralelismo se utiliza para hacer referencia a la idea de ejecución con un cierto grado de simultaneidad. Si se le llama componente a un fragmento de un algoritmo, determinada por la función que realiza dentro del mismo, entonces se le puede llamar componente paralela a aquellas componentes cuya ejecución no depende de otra. Las componentes paralelas son potencialmente ejecutables en paralelo, es decir que pueden ser ejecutadas por procesadores diferentes.

Un algoritmo paralelo es un algoritmo compuesto por componentes paralelas relacionadas, utilizando momentos de comunicación y de sincronización.

La comunicación se define como, la transferencia de datos entre dos o más procesadores y la sincronización no es más que la acción de establecer un punto en el flujo de ejecución de dos o más procesadores, en el cual cada procesador esperará por los demás, y una vez que todos (los involucrados en la sincronización) alcancen este punto, se continúa la ejecución. Cuando una sincronización involucra a todos los procesadores de una máquina paralela, se le denomina sincronización global.

9.4 ENTORNOS DE COMPUTACIÓN

En el pasado, las diferentes comunidades científicas se enfrentaron a la necesidad de solucionar problemas robustos en ingeniería, los cuales podían ser resueltos por medio del uso de grandes máquinas. Sin embargo, la consecución de este tipo de infraestructura alcanzaba enormes costos de tipo financiero.

Para resolver diversas dificultades de tipo económicas que se encuentran asociadas directamente al uso de los supercomputadores se han comenzado a utilizar soluciones más rentables, tal y como lo es, el uso de diversos entornos de ejecución encargados de repartir grandes capacidades de poder computacional a lo largo de un periodo de tiempo. Estos entornos de ejecución es lo que hoy en día se denomina “Entornos de computación”. Dentro de los entornos de computación hay que distinguir entre diversos tipos [LIARTE, 2007].

9.4.1 ENTORNO HPC

Según [LIARTE, 2007], el término HPC (High Performance Computing) se refiere al uso de supercomputadores (paralelas) y clústers de máquinas, es decir, sistemas computacionales formados por múltiples procesadores (normalmente de gran potencia), unidos en un sistema único, con conexiones comerciales disponibles. Esta arquitectura entra en contraste con las máquinas del tipo Mainframe, que son generalmente únicas por naturaleza. Debido a su flexibilidad, poder, y bajo costo, los sistemas HPC han dominado de manera incremental el mundo de los supercomputadores. Normalmente, los sistemas de computadores cercanos a la región de los tera-flops, son considerados computadores HPC.

En general los sistemas HPC son usados para investigaciones científicas principalmente.

9.4.2 ENTORNO GRID

De [LIARTE, 2007], el Grid Computing, es la tecnología que consta de una infraestructura que permite el acceso y procesamiento concurrente de un programa, entre varias entidades computacionales independientes, que actúan como un único gran sistema. Se usa normalmente para programas que requieren procesos de gran escala y/o acceso a mucha cantidad de datos.

Entre las características principales que distinguen al Grid Computing se pueden citar las siguientes:

- ✓ Posibilidad de integrar sistemas y dispositivos heterogéneos, pues permite que recursos diferentes puedan interactuar entre sí.
- ✓ Mejora del costo efectivo de los entornos operativos, pues permite aprovechar al máximo los recursos disponibles en una red, y de esta manera mejorar la capacidad de los recursos para responder a las fluctuaciones de la demanda.
- ✓ Las tecnologías grid son flexibles, pues son capaces de ajustarse dinámicamente a los entornos cambiantes y fluctuantes de las tecnologías de la información.
- ✓ Aumenta la fiabilidad de la infraestructura, sacando ventaja de los recursos del grid como una alternativa ante la recuperación de los desastres tradicionales.

Los objetivos que persigue el Grid Computing para una empresa u organización se citan a continuación:

- ✓ Mejorar los tiempos de producción: Permite incrementar la productividad y colaboración; y de esta manera las organizaciones mejoran sus tiempos de resultados y por lo tanto rapidez en el tiempo de lanzamiento al mercado, que en última instancia constituye una ventaja competitiva.
- ✓ Permitir la colaboración y promover flexibilidad operacional: No solo unirá recursos tecnológicos dispares, sino también gente y aptitudes; permitiendo de esta manera la posibilidad de compartir, acceder y gestionar información, mejorando la colaboración entre unidades empresariales.
- ✓ Escalar para satisfacer demandas variables del negocio: Permite crear infraestructuras operativas flexibles y resistentes, que faciliten abordar rápidas fluctuaciones en la demanda, accediendo instantáneamente a recursos de computación y datos para "sentir y responder" a las necesidades de negocio.
- ✓ Incremento de la productividad: Dando a los usuarios finales acceso a los recursos de computación, datos y almacenamiento que necesiten y cuando los necesiten, ayudando a las empresas a equipar mejor a sus empleados para efectuar sus tareas, resolver problemas comerciales complejos con facilidad y moverse entre etapas del diseño de productos, proyectos de investigación y más, todo más rápidamente.
- ✓ Aprovechar inversiones de capital existentes: Maximizar la utilización eficiente y productiva de los recursos existentes es una de las claves para reducir costes operativos. Además, las empresas pueden aprovechar los recursos grid para entregar

escenarios de back up y recuperación efectivos y de bajo costo, sin necesidad de invertir para duplicar sistemas.

9.4.3 ENTORNO HTC

Según [LIARTE, 2007], el término HTC (High-Throughput Computing) se refiere a los entornos que están formados principalmente por una gran cantidad de equipos de escritorio conectados en red, con el mismo o distinto sistema operativo. El objetivo de estos entornos es aprovechar los ciclos de procesamiento de los computadores cuando no se detecta actividad en ellos para poder ejecutar tareas previamente lanzadas por un usuario a un nodo central de ese entorno. Esta máquina central se encarga de monitorizar la actividad del resto de las máquinas repartiendo las tareas que le llegan de las máquinas que no presentan actividad durante un periodo de tiempo. De esta forma se aprovechan los ciclos de computación de aquellas redes que, con los computadores encendidos, no presentan ninguna actividad, para poder utilizarlos emulando un "supercomputador". Se deduce pues que la solución de un entorno distribuido HTC no es una tarea fácil de implementar debido a su complejidad a nivel software pero conlleva un gran ahorro económico.

La clave de los entornos HTC es que aprovechan el uso de todos los recursos disponibles. Existen muchas ventajas para un entrono HTC, algunas de las cuales se citan a continuación:

- ✓ HPC puede ser usado para soportar decisiones o aplicaciones de tiempo limitado. Sin embargo para hacer análisis altamente sensibles, estudios o simulaciones que necesitan para su ejecución largos períodos de tiempo, se necesita un mecanismo como HTC.
- ✓ El software que opera con un mecanismo de alto rendimiento computacional HTC, en lugar de HPC, es capaz de organizar las máquinas en clústers o colecciones de clústers.

9.5 SISTEMAS DE ENCOLAMIENTO DE TAREAS

Hoy en día existen algunos sistemas que se encargan por sí solos de balancear la carga de trabajo dentro de una arquitectura cluster o una arquitectura grid. Estos sistemas tienen como fin, agrupar determinada cantidad de tareas y posteriormente distribuirlas de la mejor forma posible logrando un buen funcionamiento del sistema.

A continuación se estudiarán tres (3) sistemas de encolamiento bastante usados en la actualidad: Condor, Mosix y Sun Grid Engine (SGE).

9.5.1 CONDOR

Condor es un software que crea un entorno de computación HTC formado por estaciones de trabajo UNIX conectadas en red. Fue creado por la University of Winsconsin-Madison (UW-Madison) a principios de los años 90s. Como otros sistemas de colas, Condor provee un sistema de encolado de trabajos, políticas de planificación, prioridades, monitorización y control de recursos. Los usuarios envían sus trabajos al planificador, que les asigna una cola y elije cuándo y dónde ejecutarlos en base a una política preestablecida, monitoriza su progreso y finalmente informa a los usuarios de su finalización. Como todo sistema HTC lo que se busca es obtener un gran rendimiento a largo plazo [VERDEJO, 2009].

De [LIARTE, 2007], los usuarios lanzan sus tareas en Condor, Condor pone la tarea en la cola, las ejecuta, e informa a los usuarios del resultado. Estos sistemas, normalmente operan sólo en máquinas dedicadas. Con frecuencia, estas máquinas pertenecen a una organización, y sólo se dedican a la ejecución de tareas para dicha organización. Condor puede fijar tareas en máquinas dedicadas.

A pesar de los sistemas conjuntos normales, Condor también está diseñado para usar máquinas no dedicadas para ejecutar tareas. Indicando que sólo ejecuten tareas en máquinas que no están siendo usadas, Condor puede de manera efectiva usar máquinas que estén en espera en un conjunto de máquinas. Esto es importante, porque a menudo la capacidad de cómputo representado por la adicción total de todas las estaciones de trabajo no dedicadas a través de un Pool (conjunto de máquinas que conforman el sistema), es mucho mayor, que el poder de cómputo de un recurso central dedicado.

El sistema Condor fue desarrollado por Miron Livny para la plataforma Linux. Nació para aprovechar el tiempo de inactividad de las máquinas, permitiendo agregar y compartir recursos, transformándose en un sistema completo de gestión de los mismos

En general las organizaciones no tienen los recursos ni la experiencia para montar un cluster que permita obtener alto desempeño de los procesos o aplicaciones que en ocasiones son necesarias.

Según [México Extremo, 2008], el proyecto Condor atiende en parte esta necesidad al generar un “sistema de encolamiento en batería a través de múltiples equipos” para procesar las peticiones; o puesto de forma más simple, se genera una red de procesamiento distribuido, de manera que una tarea determinada se procese en múltiples equipos sin interferir con la operación del usuario, lo que de alguna manera optimiza la utilización de los equipos y mejora la respuesta de las tareas.

El sistema parte del siguiente principio: Para cada equipo involucrado del proceso se instala un cliente y deberá existir un servidor que administre las tareas, que asigne las prioridades y los tiempos de ejecución, entre otras cosas. Cabe mencionar que está orientado a

Unix/Linux, aunque es posible que con Cywin eventualmente se pueda hacer trabajar en Windows.

De [NIEeS, 2006], aunque la gama de los servicios prestados por Condor es amplia y variada, se puede simplificar su funcionalidad en las tres categorías siguientes:

- ✓ Asignación de tareas: El scheduler de Condor es responsable de gestionar las solicitudes de trabajo de ejecución. Consta principalmente de archivos de entrada, ambientes de requerimientos y otros parámetros de múltiples usuarios, con el fin de mantener diversas tareas en lista de espera (cola). Los usuarios pueden asignar diferentes prioridades a sus tareas y especificar restricciones en el flujo de trabajo entre tareas.

- ✓ Servicios de administración de recursos: Un nodo central es el responsable de recolectar las características del recurso e información útil de las máquinas que forman el sistema Condor. Se basa en la información recopilada y en las prioridades del usuario, además las peticiones de trabajo pueden ser ajustadas a los recursos adecuados para su ejecución.

- ✓ Administrador de la ejecución de tareas: Condor es capaz de ejecutar tareas remotamente, suministrando mecanismos de transferencia de archivos que habilitan los archivos requeridos por la máquina remota. Los mecanismos de verificación permiten guardar el estado de la tarea para posteriormente continuar con la migración de tareas de una máquina a otra. Condor permite las llamadas al sistema realizadas por la aplicación ejecutada remotamente para ser realizada por la máquina del cliente, aportando un cierto grado de transparencia con respecto a la ejecución del trabajo.

9.5.2 OPENMOSIX

De [VERDEJO, 2009], openMosix tiene su origen en Mosix, un sistema de clustering SSI (Single System Image), cuyo desarrollo comenzó en 1981 en la Hebrew University of Jerusalem. Tras diversas encarnaciones en diferentes sistemas UNIX, Mosix es portado a Linux en 1999.

Mosix es un sistema de administración orientado a la computación de alto desempeño (HPC) sobre cluster y multi-cluster en Linux. Soporta tandas de tareas y procesos interactivos. Mosix puede ser visto como un sistema operativo multi-cluster que incorpora descubrimiento automático de recursos y distribución dinámica de carga de trabajo [MOSIX, 2009].

Basado en [BARAK, 2009], sobre un cluster Mosix los usuarios pueden ejecutar las aplicaciones que crean múltiples procesos, con el fin de permitir buscar automáticamente

los recursos y distribuir los procesos entre los nodos, para así mejorar el rendimiento global del sistema sin cambiar el entorno en tiempo de ejecución de la migración de procesos.

El descubrimiento automático de recursos de Mosix proporciona para cada uno de los nodos la información más reciente acerca de la disponibilidad de sus recursos y el estado actual en el que se encuentren. Basados en esta información y en las prioridades de los nodos, el algoritmo de migración de procesos puede iniciar la reasignación de las tareas entre los nodos proporcionando así, balanceo de carga.

El descubrimiento de los recursos se realiza por un algoritmo de difusión de información en línea, que hace que cada nodo del cluster envíe la información importante. El algoritmo es basado en la difusión aleatoria de información, en la cual, cada nodo monitorea regularmente el estado de sus recursos, incluyendo, la velocidad de CPU, su carga actual, su memoria (si está libre o no), etc.

Mosix soporta procesos de migración en cluster y en multi-cluster. El proceso de migración puede ser activado ya sea de forma manual o automática.

Las migraciones automáticas son supervisadas por algoritmos en línea que tratan de mejorar continuamente el rendimiento, es decir, si un nodo se encuentra muy cargado, el algoritmo balanceará su carga con un nodo que este inactivo. Estos algoritmos son particularmente útiles para aplicaciones con recursos imprevisibles o cambiantes, o cuando varios usuarios trabajan simultáneamente en el cluster. Las decisiones de migración automática son basadas en el tiempo de ejecución y en la información suministrada por cada nodo acerca de su disponibilidad de recursos [BARAK, 2009].

El método usado por Mosix, asegura que los procesos locales y los procesos con prioridades altas puedan moverse y asignarse a un nodo que este realizando un proceso de menor prioridad. La prioridad dentro del método permite la flexibilidad de uso dentro y fuera de los nodos del cluster.

A finales de 2001 Mosix deja de distribuirse bajo licencia GPL, y es entonces cuando nace openMosix, la versión abierta de Mosix, bajo licencia full GPL2. Esto permitió que el proyecto openMosix contase con una amplia comunidad respaldándolo y contribuyendo a su desarrollo [VERDEJO, 2009].

OpenMosix es un software que permite que computadores conectados en red funcionen bajo GNU/Linux y trabajen de forma cooperativa. Es capaz de balancear automáticamente la carga entre los diferentes nodos del cluster y permite la adición o sustracción de nodos en caliente sin necesidad de interrumpir el servicio.

La carga se distribuye entre los distintos nodos atendiendo a parámetros como tipo de conexión, memoria disponible y velocidad de CPU.

Dado que openMosix forma parte del kernel y mantiene total compatibilidad con Linux, los programas de usuario, ficheros y otros recursos funcionarán igualmente sin cambio alguno. El usuario final no nota diferencia alguna entre ejecutar sus aplicaciones en su sistema Linux o en un cluster openMosix. Para ellos, la totalidad del cluster se presenta como un gran sistema multiprocesador [VERDEJO, 2009].

De [CATALÁN, 2004], la forma como OpenMosix migra las tareas se puede resumir a continuación:

- ✓ Distribución de la carga: Las tareas se trasladan a la máquina con menos carga para evitar que haya sistemas sobrecargados. Se tiene en cuenta cuando se está acabando la memoria de la máquina, con el fin de que las tareas se dirijan hacia donde haya mayor cantidad de recursos. En el caso más óptimo se tendría la potencia de todos los nodos sumados, pues con un balanceo perfecto podrían tenerse a todos los nodos trabajando a máximo rendimiento, aunque el trabajo original se produjese en un solo nodo.
  Por supuesto este es el caso ideal y no se da en la realidad pues siempre se necesita potencia de cálculo para enviar y recibir las tareas así como para tomar las decisiones.

- ✓ Rendimiento de las comunicaciones: Las tareas que necesitan entrada/salida en un nodo se desplazan a éste para evitar traer todos los datos a través de la red. Otra variante es mantener a todas las tareas que interactúan fuertemente entre ellas en una misma máquina.

De [VERDEJO, 2009], el algoritmo de balanceo de carga migra los procesos entre los distintos nodos de forma transparente, intentando optimizar su utilización en todo momento, si bien el administrador puede modificar manualmente este algoritmo.

El hecho de que la migración de procesos se haga de forma transparente hace que el conjunto del cluster se muestre como un gran sistema multiprocesador (SMP) con tantos procesadores disponibles como el sumatorio de los procesadores de todos los nodos.

9.5.3 PORTABLE BATCH SYSTEM (PBS)

De [ABAD, 2003], PBS es un sistema de colas desarrollado en sus comienzos por Veridian Systems para la NASA, siendo su actual empresa propietaria Altair. Posee dos versiones: una de pago, llamada PBSPro y una segunda de libre distribución, denominada OpenPBS.

Existen varias versiones de este sistema, acá se estudiará la versión OpenPBS v2.3.

La configuración del sistema se llevará a cabo necesariamente desde la línea de comandos. Se configurará el servidor, los nodos de ejecución, las colas, etc. Para llevar a cabo el control del sistema: usuarios, recursos y trabajos, el administrador posee una herramienta

gráfica de acceso exclusivo. Dada la poca manejabilidad de esta herramienta gráfica, la administración del sistema de colas se lleva a cabo como la configuración, a través de la línea de comandos, lo cual se puede llegar a hacer tedioso en algunos momentos. En cuanto a la documentación aportada es bastante escasa [ABAD, 2003].

El manejo del sistema por parte del usuario se puede hacer de dos formas distintas: desde la línea de comandos o a través del entorno gráfico que se proporciona con el paquete. La sumisión de trabajos se hace a través de macros, shell scripts, en las que se especificarán tanto las características del trabajo, como los recursos que necesitará para su ejecución. Una vez comenzada la ejecución, el usuario podrá controlar (borrar, priorizar, cambiar recursos asignados, etc.) sus trabajos con libertad. La información relativa a la ejecución (tiempos, utilización de recursos, etc.) se obtiene vía e-mail. Los resultados del trabajo pueden ser redireccionados a un fichero de salida.

Se trata de un sistema de distribución gratuito donde se ofrece el código fuente. Opera en entornos de red UNIX multiplataforma, incluyendo clusters homogéneos y heterogéneos de estaciones de trabajo, supercomputadores y sistemas masivamente paralelos. No necesita ni hardware ni software adicional, no se requieren sistemas de ficheros en red como AFS o NFS, ya que las transferencias se realizan vía rcp o scp.

De [ABAD, 2003], la primera peculiaridad de este sistema es que adjudica un nodo en exclusiva a cada trabajo (spacesharing), independientemente del número de procesadores que existan o de la cantidad de memoria disponible. Si el trabajo es paralelo, el usuario debe especificar cuantos y que tipo de nodos se requieren, esto se hace a través de una herramienta mediante nodos remotos. Sin embargo, esta herramienta es muy poco utilizada, ya que los programadores en PVM y MPI prefieren usar las utilidades de las propias librerías. Una segunda característica es la posibilidad de hacer checkpoint, pero en este caso de forma muy limitada, ya que sólo se contempla en sistemas IRIX 6.5 o superiores, en este caso el checkpoint es a nivel kernel, por lo que no es exactamente el sistema de colas quien lo lleva a cabo. Por otra parte, en PBS no se contempla la migración de procesos, pero si el balanceo de carga que se hace de forma automática si se especifica en la configuración. En cuanto a la tolerancia a fallos, el sistema es bastante limitado; los trabajos que se están ejecutando en un nodo que se cae, se pierden. Por otro lado, una caída en el nodo servidor (donde residen los daemons del sistema) hará imposible la recuperación de los trabajos que estaban en ejecución. La distribución viene con diferentes políticas de scheduling, y se ofrece la posibilidad de crear nuevas políticas. La que se instala por defecto es FIFO.

De [VERDEJO, 2009], PBS consta de cuatro componentes:

- ✓ Comandos: PBS proporciona una serie de comandos en línea así como un interfaz gráfico, ambos conforme al estándar POSIX 1003.2d. Estos comandos permiten enviar, monitorizar, modificar y borrar trabajos.

- ✓ Servidor de tareas: Es la parte central de PBS. Todos los comandos y daemons se comunican con el servidor. La principal función de éste es proveer los servicios batch

básicos como recibir y crear un trabajo, modificarlo, protegerlo de posibles caídas del sistema y ponerlo en ejecución.

- ✓ Ejecutor de tareas: Es el daemon que realmente pone el trabajo en ejecución. Su nombre, mom, se debe a que es la madre de todos los procesos en ejecución. Mom pone en ejecución el trabajo cuando recibe una copia del servidor.
  También reproduce la sesión del usuario propietario del trabajo (shell, .login, .csh, etc). Finalmente, al acabar el trabajo, devuelve la salida al usuario.

- ✓ Administrador de tareas: Este daemon contiene las políticas que controlan qué trabajo, cuándo u dónde se ejecutan los trabajos.

9.5.4 SUN GRID ENGINE (SGE)

De [ABAD, 2003], SGE es un sistema de colas financiado por Sun Microsystem y desarrollado por CollabNet. Se trata de un sistema de distribución gratuita que puede ser utilizado en un gran rango de computadores distribuidos, desde un cluster hasta un sistema Grid (de ahí su nombre).

Existen varias versiones de este sistema, acá se estudiará la versión GridEngine v5.3.

Este sistema de colas posee una herramienta gráfica bastante potente, completa, intuitiva y fácil de manejar que permite al administrador llevar a cabo todas sus tareas, desde la configuración del sistema, hasta el control de usuarios, trabajos y recursos. En cuanto a su funcionalidad, es muy similar a la proporcionada por LSF.

Se puede utilizar a través de la línea de comandos, o mediante herramientas gráficas, y los trabajos o tareas deben ser sometidos en forma de shell script. La sumisión y el control de trabajos se hace de forma similar y la obtención de resultados se realiza mediante la redirección a ficheros de salida. El sistema proporciona información adicional sobre la ejecución vía e-mail [ABAD, 2003].

Este sistema es de distribución gratuita y tiene disponible su código fuente. Soporta múltiples plataformas UNIX: IRIX, Linux, Solaris, HP UX, etc. Como gran inconveniente es que para su correcto funcionamiento se requiere la presencia del sistema de ficheros en red NFS.

Este sistema cuenta con una peculiaridad especial en el soporte de aplicaciones paralelas, ya que habrá que definir y configurar unas pautas de actuación que se denomina “parallel environment”. Una vez definido este entorno, el sistema no tendrá ninguna dificultad en controlar este tipo de trabajos. Además, GridEngine soporta dos niveles de checkpoint (capacidad de suspender y reanudar trabajos): nivel de kernel (si el sistema operativo lo

permite) y nivel de usuario, con la necesidad de recompilar la aplicación. También es capaz de realizar migración de procesos entre máquinas con el fin de realizar el mejor balanceo de carga posible. En cuanto a las estrategias de planificación, será tarea del administrador configurarlas, siendo la que se implementa por defecto una estrategia de balanceo de carga [ABAD, 2003].

Es un sistema altamente configurable, sencillo de de administrar y de utilizar por parte de los usuarios. Su escalabilidad lo ha llevado a ser el sistema de clustering de mayor éxito entre los mayores clusters de supercomputación.

Como sistema de gestión de colas, SGE se encarga de aceptar, planificar y ejecutar grandes cantidades de trabajos. También se encarga de la gestión y planificación de recursos distribuidos como procesadores, memoria, disco y licencias de software.

Un cluster Sun Grid se compone de un nodo maestro y uno o más nodos de ejecución. Además pueden configurarse múltiples shadow hosts como hot spares capaces de tomar el control del master host cuando falle, proporcionando alta disponibilidad.

Al igual que Condor y PBS, Sun Grid trabaja con shellscripts que definen los requisitos de los trabajos de los usuarios, pero además puede tratar directamente con binarios e incluso ejecutar trabajos interactivos.

Se trata, sin lugar a dudas, del producto más completo disponible en el mercado, tanto por rendimiento como por facilidad de uso y capacidad de configuración. A todo esto hay que sumar el hecho de que se trata de un proyecto opensource con una comunidad de desarrolladores y administradores ingente y, algo de lo que pecan otros proyectos, el respaldo de una empresa del prestigio como Sun Microsystems.

De [SUN, 2007], la forma como Sun Grid Engine funciona se puede resumir como sigue a continuación:

- ✓ Se usa una expresión algebraica simple para encontrar un valor de carga significativo de todos o parte de los parámetros de carga reportados para cada máquina.

- ✓ El administrador define los valores de carga y la disponibilidad de recursos en todas las máquinas.

- ✓ Después de que se calcula la expresión para cada máquina las tareas son asignadas a los equipos dependiendo los valores de carga significativa. Las máquinas se organizan de acuerdo a su capacidad de carga.

- ✓ La carga es impuesta a cada máquina por SGE, y el tiempo de ejecución de las tareas varía con el tiempo, por el hecho de que cada máquina requiere de una cierta cantidad de CPU para ejecutar su trabajo.

- ✓ Cada máquina posee un umbral que determina la cantidad de trabajo que realizará, que esta denotado por la media del parámetro de carga de la CPU.

- ✓ Se ajusta un tiempo límite para que las máquinas realicen su trabajo, de no ser así el trabajo (tareas) que no han realizado será migrado a las máquinas que si hayan conseguido este tiempo, esto se hace con el fin de obtener resultados razonables.

- ✓ Usan como parámetros de cada máquina, el tiempo de CPU, la memoria y el espacio de nombres System IO.

9.6 NETWORK SIMULATOR 2

De [HERRERA, 2004], El Network Simulator es un simulador discreto de eventos creado por la Universidad de Berkeley para modelar redes de tipo IP. En la simulación se toma en cuenta lo que es la estructura (topología) de la red y el tráfico de paquetes que posee la misma, con el fin de crear una especie de diagnostico que muestre el comportamiento que se obtiene al tener una red con ciertas características.
Trae implementaciones de protocolos tales como TCP y UDP, que es posible hacerlos comportar como un tráfico FTP, Telnet, Web, CBR y VBR. Maneja diversos mecanismos de colas que se generan en los routers, tales como DropTail, RED, CQB, algoritmo de Dijkstra, etc.
Actualmente, el proyecto NS es parte de VINT proyect que desarrolla herramientas para visualizar los resultados de una simulación (por ejemplo, una interfaz gráfica).
La versión con que fue probado (en este informe) es la NS versión 2 escrita en los lenguajes de programación C++ y OTcl.
El funcionamiento de Network Simulator se explicará poco a poco mostrando las partes más generales a las más particulares. Para comenzar se mostrará una vista bastante simplificada de lo que es NS.

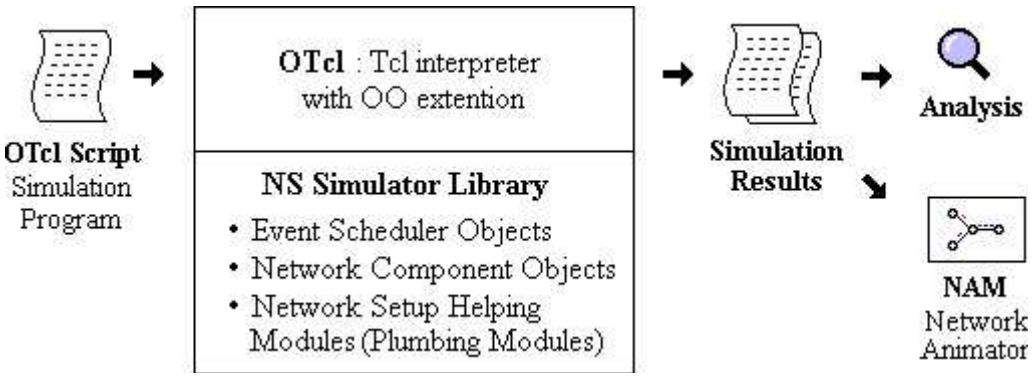

Figura 36. Vista simplificada del funcionamiento de NS

Como se puede observar, se comienza con un script en OTcl que viene a hacer lo que el usuario codifica para simular. Es el único INPUT que da el usuario al programa. El resto es el procesamiento interno de NS. La simulación queda en un archivo que puede ser bastante incomodo de leer o analizar para el usuario, sin embargo, usando una aplicación especial se puede mostrar mediante una interfaz gráfica.
El script es un archivo escrito en Tcl orientado a objetos, es decir, OTcl, que tiene diversos componentes internos que se muestran en el cuadro del medio de la figura 36. En estos componentes se configura la topología de la red, los temporizadores de los eventos, se cargan las funciones necesarias para la simulación, se planifica cuando iniciar o terminar el tráfico de un determinado paquete, entre otras cosas [HERRERA, 2004].

Como herramienta, el network simulator, es muy potente dentro del campo de la simulación de redes, y a la vez muy flexible dada la posibilidad de trabajar con scripts tcl que permiten agregar toda la potencia de un lenguaje de programación a los propios elementos de la simulación [TUREGANO, 2002].