LAN 상의 유휴 PC 들을 사용한 파이프라인 방식의 PC Cluster 의 설계

김영균*, 오길호**
*금오공과대학교 컴퓨터공학부
**금오공과대학교 컴퓨터공학부
e-mail : ygkim@cespc1.kumoh.ac.kr

Design of a Pipelined PC Cluster using Idle PCs on LAN

Young-Gyun Kim*, Gil-Ho Oh**
*School of Computer Engineering, Kum-Oh National Institute of Technology
**School of Computer Engineering, Kum-Oh National Institute of Technology

요 약

본 논문에서는 LAN 상에서 유휴 PC 들을 연산에 활용하는 PC Cluster 시스템에 대해 연구하였다. 특히, PC 실습실에 있는 PC 들의 유휴시간(Idle time)대를 이용하여 Cluster 연산에 사용함으로써 별도의 전용 클러스터 시스템을 설치하기 위한 하드웨어 및 설치 공간이 필요로 하지 않는다는 장점을 갖는다. PC 실습실의 PC 들은 주간에는 주로 교육 및 실습에 사용되며 오후 6 시부터 오전 9 시까지의 실습에 사용되지 않는 유휴시간을 CPU-Intensive 한 작업들을 병렬로 수행하는 PC Cluster 로 구성하여 저가격의 고성능 시스템을 구축할 수 있다. 그리고 특정 연산을 전담하는 노드들을 지정하고 이 노드들의 연산 결과를 인접한 다른 노드들에게 전달함으로써 연속적인 다음 연산을 적용할 수 있도록 파이프라인(Pipeline) 형태로 구성한다. 파이프라인 형태의 PC Cluster 에서 연산을 겹침(Overlapped)으로서 처리량(Throughput)을 높일 수 있다. LAN 으로 연결된 PC 실습실의 PC 들은 인터넷상의 연산 자원들보다 안정되고 신뢰성이 있기 때문에 복잡한 보안 기법을 사용하지 않아도 된다. 또한 연산시간이 유휴시간으로 고정되어 있기 때문에 네트워크의 부하 및 노드의 부하를 고려하는 복잡한 부하균등화 기법이나 스케쥴링 기법이 필요로 하지 않는다.

1. 서론

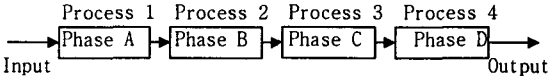
최근 PC Cluster에 대해 활발한 연구가 진행되고, 저비용 고성능의 컴퓨팅 플랫폼에 대한 해결책으로 사용되고 있다 [1,2,3]. 본 논문에서는 LAN으로 연결된 PC실습실의 유휴 PC들을 연산에 활용하는 PC Cluster에 대해 연구하였다. PC 실습실의 PC들은 주간에는 실습에 사용되다가 야간에는 사용되지 않는 유휴시간을 갖는다. 특히, 이 유휴시간에는 LAN의 네트워크 부하가 거의 없으며, PC들의 지역 작업 또한 없다. 따라서 네트워크 부하와 지역작업의 부하를 고려할 필요가 없다는 장점을 갖는다. PC실습실의 PC가 사용되지 않는 유휴시간은 평일인 경우 야간 및 심야 시간대, 토, 일요일인 경우 100% 유휴시간을 갖는다. 대학의 경우, 하계 및 동계 방학 중 연중4개월의 기간을 100%유휴시간을 가진다. 이렇게 활용되지 않는 유휴시간을 많은 연산을 필요로 하는 병렬처리 작업에 클러스터 시스템으로 구성함으로써 전용의 클러스터를 구축하기 위한 하드웨어를 구입할 필요가 없으며, 전용의 설치 공간도 필요로 하지 않는다. 각각의 PC들을 비슷하거나 동일한 성능을 갖도록 몇 개의 소규모 그룹으로 나누어 클러스터링 하고, 각 소그룹의 클러스터링간에 처리 결과를 연속적으로 다음 클러스터링 소그룹으로 전송함으로써 병렬처리가 필요로 하는 작업들의 연산을 겹침(Overlapped)으로서 전체 처리량(Throughput)을 높이는 Pipelined Clustering방법에 대해 연구하였다. 제안한 시스템은 연속적으로 입력되는 입력 데이터에 대해 단위 시간당 연산 결과를 출력하는 특징을 갖는다.

2.관련연구

2.1 PC Cluster

높은 비용의 전용 대형컴퓨터를 사용하는 것보다 낮은 비용의 PC들을 네트워크로 연결 함으로서 비슷하거나 훨씬 뛰

어난 성능을 발휘할 수 있는 클러스터(Cluster)시스템들에 대한 연구가 활발히 진행되어 왔다[1,2,3]. 클러스터 시스템은 전용의 워크스테이션으로 구성된 경우와 비전용의 PC 클러스터로 구성된 경우, 이 두 가지 형태가 혼합된 형태가 있다. 네트워크 상의 분산된 컴퓨팅 자원을 효율적으로 사용하기 위해 최근 많이 연구되고 있다.



< 그림 1. Data Pipeline의 구조 >

2.2 Pipelining

전통적인 Pipeline기법은 그림1과 같이 Fine-grained에 좀 더 가까운 병렬처리 기법으로서 기능분할(Functional decomposition)접근방식에 기반하고 있다. 각 프로세서(Processor)는 전체 알고리즘의 일부분을 수행한다.

파이프라인 기법은 기능분할(Functional decomposition)접근 방식 중 가장 단순한 형태로 널리 사용되는 방법이다. 파이프라인의 프로세스(Process)들은 파이프라인의 특정 한 단계(Stage)로 수행되고, 특정 연산을 수행한다. 통신형태는 파이프라인의 인접한 단계들간에 자료의 흐름(Data flows)으로 이루어지기 때문에 아주 단순하고 비동기적이다. 파이프라인 접근 방식의 효율은 파이프라인의 단계들(States)간에 부하를 균형 있게 맞출 수 있는 능력에 직접적으로 의존한다. 주로 화상처리(Image processing) 응용분야에서 흔히 사용되고 있다[1,7].

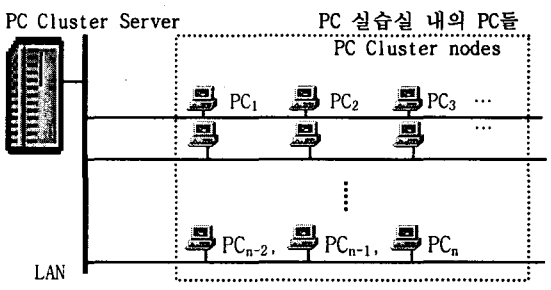
2.3 Self-Scheduling Schemes

Master-Slave Model에서의 작업 할당시 기존 분산처리 방법은 Master가 Slave Model의 작업 부하를 고려하여 Master주도로 작업을 할당하는 방식이었다. 이러한 방법은 동기화가 용이한 Master-Slave Model에 기초한 소규모의 프로세스로 구성된 전용의 클러스터 시스템을 구성할 경우 효율적일 수 있으나, 동기화가 어려운 서로 다른 작업 시간을 갖거나, 서로 다른 성능을 갖는 노드로 구성된 경우 Master주도형의 작업 스케쥴링 방식은 비효율적이다. 따라서 서로 다른 성능을 가질 경우 Slave가 유휴 상태일 때 Master에게 작업을 요청하는 방법인 Self-Scheduling[4]방법을 사용하는 것이 보다 효율적이다.

3.시스템의 설계

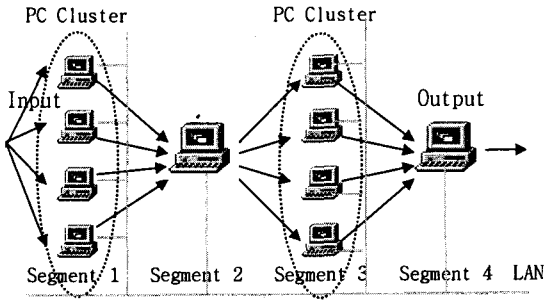
3.1 시스템의 구성

제안한 방법을 사용하는 시스템은 TCP/IP 프로토콜을 사용하는 LAN환경의 PC실습실내에 위치한 PC들의 유휴시간을 활용하는 것으로 그림1과 같이 물리적으로 구성 되었다

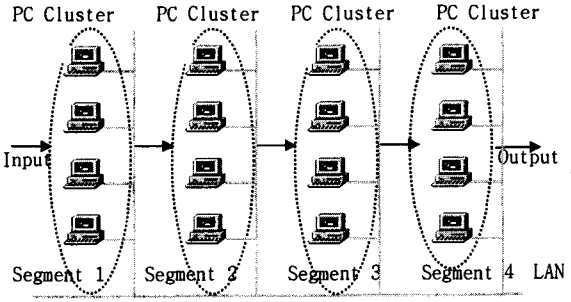


<그림1.실습실 PC들을 사용하는 PC Cluster의 물리적 구성>

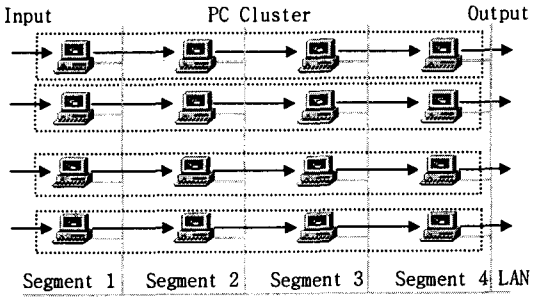
그림 2의 a)는 뛰어난 성능의 소수의 PC와 성능이 저조한 다수의 PC들로 구성된 경우로서 Segment를 구성하는 Cluster가 서로 다른 성능의 PC로 구성된 경우이다. 그림 2의 b)는 동일한 성능의 PC들을 동일한 크기로 Pipeline의 각 Segment를 구성한 경우로서 Segment의 성능이 동일하다. 그림 2의 c)는 서로 다른 데이터 집합에 대해 Segment별로 서로 다른 코드를 수행하는 경우이다. 이 경우 4개의 Segment로 구성된 Pipeline이 4개 있는 경우로 단일 Pipeline에 비해 처리량이 4배가 된다. 그림 2의 d)는 Segment간에 작업의 크기로 서로 다른 작업을 배치한 경우 Segment간에 연산 속도차이를 완화하기 위해 하드 디스크의 일부를 앞 Segment의 연산 결과를 저장하기 위한 Cache로 사용하는 경우이다. 그림 2의 e)는 초기에 동일한 크기의 Segment로 구성되어 있다가 최초의 출력이 나오기까지의 각 Segment간의 연산결과 시간을 측정하여 두번째 연산에서 각 Segment간의 연산 시간을 균등하게 하기 위해 동적으로 특정 노드를 다른 Segment로 구성 시키는 경우에 Segment간 부분 연산결과의 통과 시간을 사용하는 경우이다.



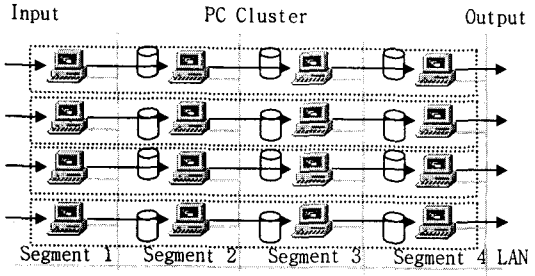
a)서로 다른 성능의 PC를 Pipelined PC Cluster구성한 경우



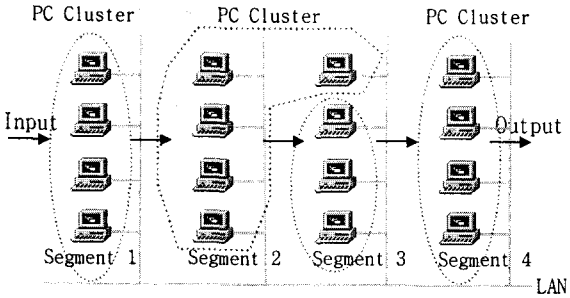
b)동일한 성능의 PC를 Pipelined PC Cluster로 구성한 경우



c) MIMD형의 Pipelined PC Cluster로 구성한 경우



d) Pipelined PC Cluster의 Segment간에 하드 디스크의 일부 공간을 Cache(Buffer)로 둔 경우



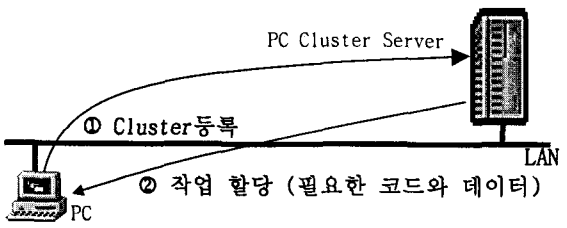
e) Pipelined PC Cluster의 Segment간에 부하에 따라 Segment의 크기를 동적으로 재구성 한 경우 (이 경우 2번 Segment에 노드를 늘려 주었다.)

<그림 2. 제안한 Pipelined PC Cluster시스템의 구성>

3.2 작업할당

PC실습실의 PC들을 유휴시간에 응용분야별로 적절하게 논리적으로 재구성함으로써 새로운 응용에 하드웨어의 구성을 변경하지 않고 쉽게 적용할 수 있다는 것이 실습실 PC Cluster의 큰 장점 중 하나이다. 유휴시간에는 LAN의 사용자가 없기 때문에 네트워크 부하가 거의 없고, 각 클러스터를 구성하는 PC들의 지역 작업이 없기 때문에 동적인 부하 균등화 기법을 사용할 필요가 없다. 따라서 정적으로 스케쥴링된 작업을 배치하여 연산이 가능하므로 스케쥴링 알고리즘이 동적인 방법에 비해 단순하다는 특징을 갖는다.

연산에 참여하고자 하는 LAN상의 노드는 1)Master에게 접속하여 Cluster연산에 참여하기 위해 유휴시간과 IP를 등록한다. 2)클러스터 서버는 노드를 구성하는 각 PC에게 연산에 필요한 코드와 데이터를 전송하여 작업할당을 수행 한다.



<그림3. 제안한 시스템의 초기 작업 할당>

3.3 동기화 문제

Pipelined PC Cluster의 Pipeline을 형성하고 있는 각 Segment의 소그룹 Cluster간의 동기화 문제가 발생할 수 있다. 전송해야 할 연산 결과의 데이터 크기가 고정되어 있는 경우 보다는 가변 크기의 연산 결과를 가지는 경우 각 Segment별로 송신과 수신에 소비되는 시간이 서로 다르게 되어 연산시작 시간이 노드 사이에 불일치할 수 있다. 데이터의 크기가 고정되어 있고 크기가 소규모인 경우는 동기로 인한 낭비되는 시간이 적지만, 연산 결과의 데이터의 크기가 커질수록 동기화로 인한 시간 낭비는 커지게 되는 문제점을 갖는다. 또한 각 Segment별로 Cluster에 할당된 작업의 연산시간이 동일하지 않기 때문에 Pipeline의 각 Segment를 구성하는 클러스터간의 동기화 문제가 발생된다.

Pipeline의 depth가 깊을수록 Pipeline의 각 Segment간에 연산결과를 전송해야 하는 시간이 전체 연산시간에 비해 커질 수 있다. 따라서 응용분야별로 적절한 크기의 depth를 결정하는 것도 중요한 문제이다. Self Scheduling기법[4]을 이용하면 각 노드 사이에 가변 크기의 데이터를 전송함으로써 발생되는 시간을 최소할 수 있다. 즉, 특정 Segment의 클러스터가 큰 데이터를 수신하는 동안에 다음 Segment를 구성하는 클러스터에 Pipeline연산과 독립된 연산시간이 짧은 작업들을 배치함으로써 특정 Segment의 클러스터를 구성하는 PC들이 앞Segment의 데이터수신과 연산종료에 걸리는 시간 동안에 유휴상태로 대기하고 있는 시간을 활용할 수 있게 된다. 그러나, 이 방법에는 조금 더 복잡한 스케쥴링 기법이 사용되어야 하고, 배치되는 짧은 연산 작업의 연산 종료 시간을 정확히 예측하기 어렵다는 새로운 문제를 갖게 된다. 따라서, 이전 Segment의 클러스터가 연산을 종료하자마자 다음 Segment의 클러스터는 현재 수행중인 작업을 중단하고 이전 Segment의 데이터를 수신하고 Pipeline연산에 참여해야 한다. 동기화 문제를 해결하기 위한 다른 방법은 Pipeline을 구성하는 각 Segment 사이에 속도차이를 완화하기 위한 Cache(Buffer)를 수신하는 쪽의 하드 디스크의 일부 공간에 구성함으로써 해결 가능하다. 그림 2의 e)처럼 Pipeline의 Segment간의 연산 속도차로 인한 성능저하를 막기 위해 시간이 많이 걸리는 연산을 담당하는 Segment의 노드 개수를 동적으로 늘려 줌으로써 Segment간의 처리 속도를 동일하거나 비슷하게 일치시켜 준다. 이 경우 초기 연산 데이터와 작업을 배치하고 연산시작으로부터 각 Segment가 연산을 종료하고 다음 Segment로 전송하기까지의 시간을 측정한다. 모든 Segment의 연산 종료시간을 비교하고 수행시간이 많은 Segment에 노드를 적절하게 추가함으로써 Segment간의 연산속도 차를 완화시킨다.

3.4 PC Cluster에 참여하는 노드의 구성

연산에 참여하는 PC들은 각각의 유휴시작시간과 유휴종료시간을 갖는다. 또한 파이프라인의 각 Segment를 형성하도록 소규모의 Cluster로 구분되어 관리된다.

<표1. Pipeline Segment를 구성하는 PC들의 등록 테이블>

No.	PipeLine Segment	IP address	유휴시작시간	유휴종료시간	할당작업
1	Segment 1	202.31.130.91	오후6시	오전8시	Work 1
...	...	...	...	...	...
N	Segment n	...	오후6시	오전8시	Work n

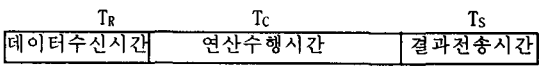
PC Cluster Server는 표1과 같이 PC실습실의 PC에 대해 Cluster에 참여하는 PC를 Pipeline의 각 Segment별로 할당하여 작업을 관리한다. 동적으로 Segment가 재구성되는 경우 PC가 참여하는 Segment를 변경함으로써 Pipeline을 재구성한다. Pipeline의 마지막 Segment에 속하는 PC들은 연산결과를 PC Cluster Server로 전송함으로써 Pipeline 주기를 완성한다.

4. 제안한 방법의 평가 및 고려 사항

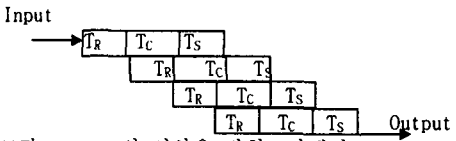
Pipelined 클러스터에서 단일 Segment의 작업 시간은 그림 4의 a)와 같이 전단계 Segment클러스터로부터 데이터를

수신하는 시간 T_R와 수신한 데이터에 대해 연산을 수행하는 시간 T_C, 그리고 연산 결과를 다음 단계의 Segment클러스터에게 전송하는 시간 T_S으로 그림 4의 a)와 같이 구성된다.

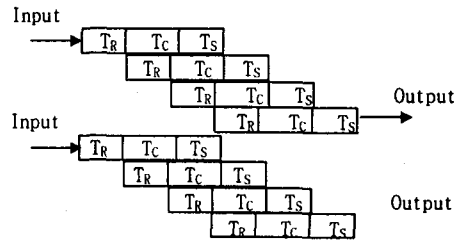
그림 4의 b)와 같이 일정시간 후에 연산결과는 매 단위시간마다 출력을 갖는다. 그림 2의 c)와 같이 구성된 경우 Pipeline이 복수개가 있게 되므로 매 단위시간마다 출력되는 결과는 그림 4의 c)와 같이 Pipeline의 개수만큼 곱해진 처리량(Throughput)을 갖는다. 따라서 처리량을 높이기 위해서는 Clustering된 Pipeline을 여러 개로 구성 한다. 또한 Pipeline을 구성하는 Segment의 크기인 |Segment|를 크게 함으로써 연산 속도를 높일 수 있다. 즉 Segment를 구성하는 PC를 늘려줌으로써 연산 속도를 높일 수 있고 Segment당 할당하는 작업의 크기를 단순화 시키고 Segment의 개수를 늘려 줌으로써 전체 처리 속도를 높일 수 있다.



a)단일 Segment의 작업 시간 구성



b)각 Segment의 연산을 겹친 4단계의 Pipelined Cluster의 작업 시간 구성



c) MIMD 또는 Superscalar형태의 다중 Pipeline을 가지는 Pipelined 클러스터의 작업시간 구성

<그림4. Pipelined 클러스터의 작업 시간 구성>

그림 4의 처리 시간에는 초기 작업을 배치하는 시간과 연산결과를 PC Cluster Sever에 전송하여 연산결과를 통합하는 시간이 제외되어 있다. 따라서 전체 처리 시간 T_{total}은 초기 작업 배치 시간을 T_{alloc}, 파이프라인 연산시간을 $T_{pipeline}$, 결과 통합시간을 $T_{Collect}$라 할 때 식1과 같이 주어진다.

$$T_{total} = T_{alloc} + T_{pipeline} + T_{Collect} \quad (식1)$$

파이프라인에서의 연산시간 $T_{pipeline}$ 은 단일 Segment의 작업시간을 $T_{segment}$, Segment개수를 $N_{segment}$ 할 때 식2와 같이 주어진다.

$$T_{pipeline} = T_{segment} \times N_{segment} \quad (식2)$$

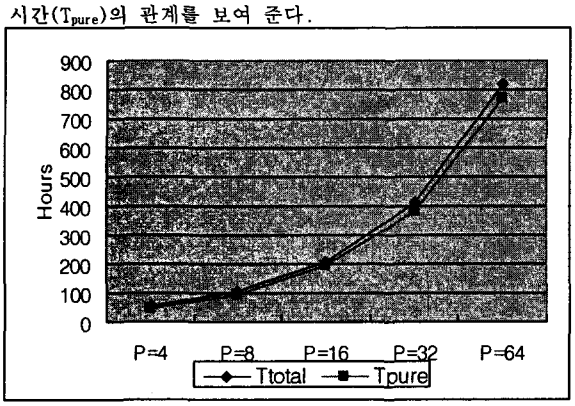
단일 Segment의 작업시간은 식3과 같다.

$$T_{segment} = T_R + T_C + T_S \quad (식3)$$

단일 Segment에 참여하는 노드의 개수를 $P_{segment}$라 할 때 $P_{segment}$로 구성된 Pipeline의 한 Segment의 연산가능시간 $T_{segment.P}$ 는 (식4)와 같이 주어진다.

$$T_{segment.P} = T_{segment} \times P_{segment} \quad (식4)$$

그림 5는 T_{alloc} =0.3, $T_{collect}$ =0.5, $N_{segment}$ =4, T_R =0.1, T_S =0.1 T_C =3 일 때 Segment간에 고정된 연산시간과 고정된 데이터 전송시간을 갖는 경우의 $P_{segment}$에 따른 Pipelined PC Cluster의 전체 연산시간(T_{total})과 순수연산에 참여하는 시간(T_{pure})의 관계를 보여 준다.



<그림5. Segment 개수가 4일 때 노드 수에 따른 전체 연산시간과 순수 연산수행 시간의 관계>

Segment간의 데이터 송수신시간(T_R ,T_S)이 Segment연산시간(T_C)보다 상대적으로 작은 경우이기 때문에 전체연산에 걸린 시간과 총 순수연산시간과의 차이가 작게 나타났다.

5.결론 및 향후 연구방향

본 논문에서는 LAN상에 있는 컴퓨터들로 구성된 Pipelined PC Cluster시스템에 대해서 연구하였다. PC를 특정 작업을 수행하기 위해 소규모로 여러 개의 클러스터로 클러스터링하고, 소규모로 클러스터링 된 시스템간에 연속적으로 계산결과를 다음 연산을 수행하는 클러스터에게 전송함으로써 처리량을 높이기 위한 방법을 연구하였다.

PC 실습실의 PC를 이용한 Pipelined PC Cluster시스템은 별도의 전용의 PC와 하드웨어를 확보할 필요가 없고 별도의 설치 공간이 필요로 하지 않기 때문에 저가격의 고성능 PC Cluster를 쉽게 구축할 수 있다. 차후, Pipelined PC Cluster의 효율을 높일 수 있는 방법에 대해 연구해 보겠다.

참고문헌

[1] Rajkumar Buyya, High Performance Cluster Computing, Vol.2, Programming and Applications, Prentice Hall PTR, 1999

[2] Putchong Uthayopas, Surachai Phaisithbenchapol, Krisana Chongbarirux, "Building a Resources Monitoring System for SMILE Beowulf Cluster", Proceeding of High Performance Computing, Asia '99, 1999

[3] Rajkumar Buyya, "PARMON: a portable and scalable monitoring system for clusters", SOFTWARE-PRACTICE AND EXPERIENCE, Softw. Pract. Exper. 2000; 30:1-17

[4] Anthony T. Chronopoulos, Razvan Andonie, "A Class of Loop Self-Scheduling for Heterogeneous Clusters", Proceedings of the 2001 IEEE International Conference on Cluster Computing(CLUSTER'01)

[5] Kam Hong Shum, "Fault Tolerant Cluster Computing through Replication", *Proceedings of the 1997 International Conference on* Parallel and Distributed Systems(ICPADS '97)

[6] Partha Dasgupta, Zvi M. Kedem, Michael O. Rabin, "Parallel Processing on Networks of Workstations: A Fault-Tolerant, High Performance Approach", 15th Intl. Conference on Distributed Computing Systems, May 1995, Vancouver, BC, Canada

[7] Jian Yang, Jiaoying Shi, Zhefan Jin, Hui Zhang, "Design and Implementations of A Large-Scale Hybrid Distributed *Graphics System*", Fourth Eurographics Workshop on Parallel Graphics and Visualization(2002)