The Memory and Communication Subsystem of Virtual Machines for Cluster Computing

Shiliang Hu & Xidong Wang
{shiliang, wxd}@cs.wisc.edu

Abstract: Virtual machines have demonstrated promising potentials in many co-designed microprocessor, uniprocessor and multiprocessor system projects for their many advantages over traditional systems, e.g. design flexibility, compatibility, performance, fault tolerance and containment. An interesting space left for us to explore is virtual machines for cluster computing environment. In this project, we studied the memory and communication subsystem of this potential new paradigm for cluster computing. We started with a dream of a complete system design draft for a virtual Machine monitor system that turns cost-effective commodity SMP systems into a highly parallel/scalable NUMA-like multiprocessor system running multiple virtual machines on top of the cluster for various purposes. However, the real tough memory system challenges and our simulation results show that this dream seems still need a lot of work and or industry changes before it becoming reality in current technology settings.

1. Introduction And Motivations

Two recent trends in the computing society made us think about studying virtual machines for cluster computing. The first trend is that dozens of virtual machine projects have demonstrated promising opportunities in many co-designed microprocessor, uniprocessor system and multiprocessor system projects for their attractive advantages over traditional systems such as design flexibility, software compatibility, performance, power and fault containment. The other trend is that the computer system construction strategy has been strongly shifting towards integrating cost-effective and technology-tracking commodity modules instead of involving large percentage of costumed design work in the whole system design and development. In particular, the commodity high speed network technologies and SMP systems has been enabling clusters having a comparable configurations with multiprocessors and multi-computers.
Meanwhile, we believe the demands for multiprocessors and/or cluster computers will keep strong in many application areas like commercial servers, scientific, engineering, graphic, vision, virtual reality or environment, entertainment and many, many other areas.

Client-Server is the most popular computing paradigm in the commercial, personal and Internet computing communities. It seems reasonable to assume that the demands for powerful high end servers will keep strong in the future market while most server owners could only have limited budgets, and variety of platforms, software bases across generations due to evolution of the companies. Facts indicate that commercial servers in these areas take the majority of today’s high-end computer market.

High performance scientific and engineering computing demands will keep increase. Traditionally the major demands in high-end computer market, they still play an important role as more and more scientific research and engineering design works are adopting simulation, which is in general computation resource intensive, as their important methods for doing research or design.

Computer graphics, vision, multimedia, all kinds of demanding virtualization applications, computer animations and their applications in entertainment, and even virtual reality could be major new applications in the future for high end machines. For example, it took Pixar Corporation several months and a lot of human resource on large Linux clusters to make the animation “Monster Inc.”. The similar situations occurred during the making of such popular movies like Titanic and Toy Story.

Although there are enough application demands for parallel and/or distributed computing, the best programming paradigm is still not very clear yet, both shared memory and message passing have their own advantages and disadvantages. Shared memory is obviously easier to programmers, but notoriously difficult for system designers to achieve in large scalable systems. In fact, in a large scalable multiprocessor or cluster system, shared memory is provided ultimately through low level message-passing through very high speed system interconnection network operated by hardware controllers or firmware, very low level software. Therefore, the problem lies in, as many architecture problems, where should we draw the interface between the upper level and lower level. Should the message passing be exposed to the ultimate programmers or users? Seems the trend is that in systems that shared memory interface could not be provided conveniently, the message passing mechanism would be naturally exposed to users. The Tempest and Typhoon paper studied some related issues on this hard problem.

A New Paradigm for Cluster Computing?
Many people believe NOW/Clusters are the most cost-effective way to build large scalable systems for high end market and it’s verified to a large extent by the industry.

There are many ways to couple SMP nodes together. For example,

1. Program applications as distributed ones with standard APIs like PVM, MPI. This demands users to port shared memory parallel programs to distributed ones.

2. Develop distributed OS to support this. Turn the burden to OS, which are always lag behind hardware development.

3. Make them NUMA like in architecture, possibly with some gluing hardware modules to provide a NUMA architecture interface to the upper layers while still having flexibility in hardware module design by means of virtual machine technology.

VMM that turns clusters into a NUMA-like multiprocessor virtual machines has the following advantages or disadvantages: It inherits large base of system and application software with little or no changes to them, thus saves a lot to the huge software base. It may provide more opportunities for architecture innovations by adding a piece of micro-software on top of hardware. The system could grow at flexible granularity, and looks more scalable than conventional multiprocessor technology in reality. Obviously, this makes extra burden to the architects and it may not actually be attractive to end users for many reasons, e.g. unpredictable performance or just bad performance due to bad locality.

Based on the work from Cellular DISCO VMM for NUMA-multiprocessors, we propose a similar work for virtual machine monitors on top of (SMP) clusters to turn cost-effective commodity SMP systems into highly parallel, scalable and cellular NUMA-like multiprocessor systems to support desired parallel virtual machine models for various purposes (Figure 1). For example, A virtual machine supports the native ISA of the node and SMP parallel programming model to execute familiar applications on the local node; A parallel virtual machine with a specific ISA and message passing API to support some message passing or cluster applications; And yet another parallel virtual machine across multiple nodes to support shared memory parallel programming paradigm.

An important property of this paradigm deserves notice is the multiplicity everywhere in the system -- multiple ISA, multiple - OS service models, API or paradigms are supported simultaneously in the system. No single entity in any single layer in the whole system, even not at the OS level, which is often the performance and reliability bottleneck in multiprocessor systems. In fact, large-scale multiprocessors are
not often used as the way they are supposed to work. OS and other important software components always lag behind and most importantly, single OS entity in the whole system decrease both reliability and scalability. By organizing each node as a cell in the whole system, we have multiple cellular VMMs running on top of cluster computers and multiple ISA/OS/parallel programming support, therefore, increase reliability, scalability, and flexibility.

There are many interesting issues/topics to study for this paradigm:

Fine grain resource sharing of common server resources such as processors, memory, network interfaces and disks among cluster nodes. For example, VCPU migration for load balance, memory sharing and parallel disk I/Os.

Check-pointing and VM/process migration for both HW and SW fault containment. In addition, we can study dynamic hardware reconfiguration via fault recovery mechanisms inside a co-designed VM.

Dynamic verification of cache coherence/shared memory coherence and memory consistency models with virtual machine support as protocols are becoming aggressive and difficult, expensive to test and verify in design and manufacture. HW alone may not be a good solution.

Search for a potentially unified solution for such VMM system for both clusters and large scale multiprocessor systems as the architectures are converging.
Finally, as there are many attractive features that we want to support in the VMM on top of clusters, e.g. multiple ISA support, dynamic verification, fault containment, dynamic configuration etc. We need to figure out some effective ways to integrate various VMM technologies without losing major VMM advantages over conventional OS kernels like reliability, fast maturity, flexibility and extensibility.

However, in this course project we focused on studying running a single virtual multiprocessor machine across a cluster of SMP nodes as the first step to reach the above attractive cluster computing paradigm. Specifically, we only study the memory and communication subsystem of such a virtual machine framework.

2. Related Works

There are many related projects on NOW or cluster computing paradigm.

The Berkeley Network of Workstations (NOW) project emphasizes on high speed commodity networking, workloads and parallel disk I/O and many other issues. They seek to harness the power of clustered machines connected via high-speed switched networks. By leveraging commodity workstations and operating systems, NOW can track industry performance increases. The key to NOW is the advent of the killer switch-based and high-bandwidth network. This technological evolution allows NOW to support a variety of disparate workloads, including parallel, sequential, and interactive jobs, as well as scalable web services, including the world's fastest web search engine, and commercial workloads.

Princeton SHRIMP, the SHRIMP (Scalable High-performance Really Inexpensive Multi-Processor) project focuses on the communication subsystem for clustered machines and investigates how to construct high-performance servers with a network of commodity PCs, and commodity operating systems. Their goal was to study how to build such a system to deliver performance competitive with or better than the commercial multi-computer servers with significantly less cost. Virtual memory mapped user level communication (VMMC) system and user level shared virtual memory systems like Lazy Release Consistency, Automatic Update Release Consistency are investigated in the project. With VMMC support, their system can write through the network on the cluster.

Rice TreadMarks studied user level shared virtual memory and sub-blocking techniques to solve the severe false sharing problem. They assume release consistency memory model and perform it in a lazy
fashion. When user code write to a page, they make a copy of the page and write data to the new page. When acquire or release primitives happen, the merge is done by do a diff of the original page and written page. Both diff info and the page are communicated for merging. The memory copy and diff operations are blamed for performance problems.

DEC-WRL worked on another system called Shasta. They provide transparent interface to existing application code by do a simple binary translation of shared memory accesses at load time. They also pinpoint out some problems with user level shared virtual memory projects carried out elsewhere. Other projects like SoftFlash and Stanford FLASH studied providing shared memory interfaces to upper application code.

There are also a bunch of high speed networking technology projects for NOW/Clusters like Berkeley Active Message, Illinois Fast Message, Cornell U-Net. Taking into account the fact that the speed and latency of commodity networking devices improve very fast, in fact even faster than the custom system interconnection designs would improve because they are cheaper and better track technology, many assumptions and designs in current systems might have become the reasons of inefficiency and cost. For example, in most current message passing system or shared virtual memory systems, communication via OS service for all user communication is an inefficient scheme. Data shows that on Intel DELTA multi-computer, sending and receiving a message requires 67 uSec, of which less than 1 uSec is due to hardware latency. The main reason for such high overheads is that communication is provided as in the old times when networking software concerned too much about the poor networking condition and protection for every user communication activity. Another comparison, user space communication with Myrinet technology has a latency of 5 - 17 uSec while the Rice Treadmarks shared virtual memory would take 500 uSec for a short message to pass to another node. The high speed and low latency of today’s high speed commodity networks provides both opportunity and challenges.

3. Architectural Design Drafts
In this report, we propose two design drafts targeting towards low end and high end market respectively. Emphasis would be on memory and communication system for both systems. Before that, let’s describe some overall concerns that should be taken in account in the designs. The draft designs are simplified and explored by SimOS-PPC multiprocessor simulator. The experimental results are presented in section 4.

Ideally, the following features are desired for high end machines:

- High performance and capacity -- Hundreds of up-to-date processors, Huge memory, permanent storage and communication capacity may be needed for many applications.
- Reliability -- The availability of the system, fault containment for both Software and Hardware faults.
- Cost Effectiveness.
- Scalability -- Including both the hardware platform scalability and the scale of applications, which may have different computing power requirement and ABI/ISA assumptions, it can support.
- Flexibility – Features like dynamic reconfiguration, platform independence, multiple ISA support and multiple parallel programming paradigms e.g. message passing or shared memory.

As for a typical server end system, architects care most about such common resources as processors, memory, and I/O devices with special interests in networking interface and disks as they are the most concerned I/O devices for most sever end workloads. We describe our concerns and problems on these components accordingly.

- **Processors.**

  This is the most mature area and technologies such as binary translation, dynamic profiling, monitoring and optimization would satisfy our multiple ISA support purpose. Other issues to be studied in future work: VCPU check-pointing and recovery, VM migration.

- **Memory**

  Shared memory placement, caching, migration and coherence protocols inside co-designed virtual machines. For example, we could use some hidden memory for shared memory caching or other purposes. Similar to NUMA-awareness in DISCO and Cellular DISCO, our VMM should be cluster-aware in its resource management. This is a kernel part of the design. We have done some simulations to study memory coherence performance, traffic and false sharing in different parameter settings. This design would tangle with memory consistency models, which is subtle and many people are not sure of its future.
One possibility is to design a FLASH MAGIC-like protocol processor as part of a co-designed VM to support shared memory among clusters in an excellent networking environment comparable to that of multiprocessors. Another possibility is at page level but with sub-blocking on a network not comparable to multiprocessors or an OS VMM is desired for no special hardware support.

**Network Interfaces.**

We argue that, just as microprocessor designs took multiprocessor issues in mind several years ago, SMP system designs could also take the inter-cluster communication issues into account for better larger system integration. Specifically, we argue that the high speed networking interface should not be put in slow system area like some current I/O levels. It might be a good idea to put high speed networking devices directly on the memory interconnection, e.g. the snooping bus or other inter-processor and memory module interconnections.

We anticipate that there might be multiple network connections in the future systems for various purposes, some high speed links for cluster computing w/o security concerns and some external links for internet access w/ much security concerns. VMM could help on these connection management provide a better the interface between the HW links and the upper OS and other software.

As many research work focus on providing high speed networking service as user space service to reduce software overhead. VMM could virtualize the networking interface in front of various user space networking paradigms.

**Disks.**

One important outcome of the Berkeley NOW project is that they observed that for NOW architecture and its targeted workloads, the most benefits doesn't necessarily come from parallel processing, but rather from sharing memory among the nodes and the fact that NOW have a great potential for doing parallel I/O in a really cheap and scalable way. As the network speed is much faster than disks, paging to another cluster node would be much better than paging to disks.

**VMM for Commodity SMP Clusters Without HW support**

Our first scheme assumes reasonable network connection at I/O level, say, switched 100Mbps or Giga-bps Ethernet, see Figure-2. We focus on VMM memory system that support page level shared memory
across cluster nodes with sub-blocking techniques to avoid severe page thrashing due to false sharing. Message passing performance could also be improved on the network by means of bypassing some OS overhead and make disks visible to multiple nodes for disk sharing and potential parallel disk I/O. No specific hardware support is assumed.

Figure 3 shows a possible way to obtain fine-grained shared memory among clusters. VMM has shadow page tables and PA to MA translation mapping. For private pages and shared read-only pages, VMM performs private local page allocation and page replication/migration respectively to improve performance. For write sharing pages, associate an thrash counter with each of them. Before the counter reaching a threshold, do page level coherence as a page-level SVM does. When the counter reaches a threshold, the corresponding page is suspected as a heavily false shared page. VMM changes the page protection bits to Read-Only and allocate cache line directory data structure for the page to track coherence states for each cache line/sub-block in the page. When a CPU writes to a sub-block in the page, the RO protection traps to VMM first, VMM checks if this is a true protection fault or a false one. The former would be reflected back to OS for handling while the latter would go through our block level coherence protocol. In particular, we can do a binary translation of the code page that touches on the suspected data page to generate a new code page in the VMM translation cache. The new piece of code in translation
cache can invoke our fine grained coherence code in VMM. Each page has a home node, write-backs go to home node. Coherence protocol messages from the network are handled by VMM handlers. VMM calls on network driver directly, avoiding OS overheads for memory coherence. Memory consistency relies on the protocol design.

There are some important issues to be considered here:

- **Kernel data and locks.** One big problem with virtual machines across cluster nodes is that if we boot OS across SMP nodes, those big kernel data structures and locks could make great troubles. And Our simulation verified this concern about those big OS kernel page tables, buffer chains/hash-tables……

- **Self-Referencing and Self-Modifying code.** Solutions for this problem are highly depending on the specific architecture under consideration. For PPC-ISA, which has separate bits for read, write and executable protection bits, we could set the new translated binary page as executable only and any further read accesses for self-referencing purpose would cause protection trap, which let VMM to guide to read to the original code page. Similarly, Self-modifying code is handled. Should we have separate ITLB and DTLB, the problem can also be solved easily. For architectures without these features, this could lead to problems.
• Flexible granularity. A potential advantage of this scheme is that we could use flexible granularity in VMM coherence code, depending on user code behavior. However, the cache line size of most CPUs with commodity cache make this flexibility difficult to exploit.

• **VMM for High End System with HW Support**

  This scheme assume user have excellent commodity high speed network connection, thus can afford to support all fine grained shared memory, high speed user space network interface, shared disk array. As most high speed network needs network processor for high performance user space communication, they ask for extra hardware. For example, TLB, Reverse-TLB and bus snooping for memory operations mechanisms in the network interface are needed in Reflective Memory, Memory Channel and SHRIMP VMMC. The hardware infrastructure that the memory system coherence protocol requires can be integrated naturally with the high speed network communication interface as the way Tempest and Typhoon did. While they did it for user level processes and asked for compiler or programmer support, VMM does it below OS and would not ask for extra support from compilers or programmers. Unified solution for shared memory and communication with HW communication assist design and VMM code could be a better solution.

  VMM can either bypass disks to upper OS for better data distribution and parallel disk I/O support by making disk resources visible across cluster nodes. We assume two major kinds of disk activities: Get some persistent data from disks and VM paging. The former could be helped by distribute data across multiple disks and having large disk buffers within the cluster. The latter could be replaced by paging to remote nodes. High speed network would help on both remote disk access and remote paging as network latency is significantly smaller than that of disk access.
4. Simulation Results

- Simulation Setup.

We carried our simulation on white lace, an 18-processor IBM RS/6000 S80 enterprise server with 24GB shared memory. We choose SimOS-PPC for simulation speed on the PowerPC ISA based IBM server. The benchmarks are Specweb99 and TPC-W due to the fact that we are currently more interested in commercial servers than other workloads. As we only have 4-processor checkpoints for the benchmarks, we focus on 4 nodes clusters in our simulation. The simulated machines has the following configuration: Total 4 nodes in cluster with 1024MB physical memory seen by the OS running on top of the cluster. For each node, the configuration is 1-processor per node, 512MB machine memory managed by VMM. Extra memory is used by VMM for page migration and replication, translation cache for binary translation, and the hidden memory needed by the VMM for whatever reason. SimOS-PPC doesn’t support NUMA or even an exact bus-based SMP memory system as the original SimOS does. We implemented our module for cluster memory system simulation and got some preliminary results. However, the numbers from the simulator looks still not quite consistent and problematic till this point and need a lot of further work on it for better simulation results.
We choose specweb99 and TPC-W as our benchmarks. The TPC-W benchmark models an on-line bookstore and is designed to exercise a web server/database system, which simulates a typical E-commerce application environment. The TPC-W benchmark intends to exercise these components with different web interactions. The TPC-W specification defines three different mixes of web interactions, Browsing Mix, Shopping Mix and Ordering Mix. Each particular mix is designed to represents a certain web surfing scenario. For example, the browsing mix would spend 95% of the web interactions on browsing related web interactions and 5% on ordering related web interactions, however, an ordering mix would tend to send more time (50% web interactions) on ordering related web interactions.

- **Cache Level Simulation Results**

![Graph showing cache level simulation results](image)

In cache level performance evaluation and analysis, we hope to discover how the design will degrade the cache level performance compared with CC-NUMA machine. False sharing will cause OS performance problems on both CC-NUMA and bus-based system. In CC-NUMA machine, this will cause additional data transfer in bus or interconnection network, while in our design, this data transfer happens in a very-high-speed network for high-end design or in a high-speed network for low design. Percentage and data volume of false sharing will influence the system performance more seriously. Therefore cache level memory access pattern should be analyzed in detail to discover as many as possible clues to improve performance.
The cache level access pattern in Specweb benchmark is carefully analyzed. Firstly from trace data of a particular process, it can be stated that most of memory accesses are absorbed by 1" level cache and 2" level cache. Local hit rate in 2" level cache is 83.21% of total memory access, and the data it accessed is 83.79% of total memory access data. That result is inspiring and it means that most of memory access only needs less than 10 process clock cycles to finish.

As to those memory access missed in local cache, they can be served either in remote cache or from global physical memory. Now we only consider the case of hit in remote cache, since in global physical memory part, the other case will be discussed in detail. There are 1488728 local misses (or 16.78%) in total, while the miss data volume is 5725805 bytes (or 16.20%). On average about 3.7 bytes is accessed in each missed memory operation. That is a very small number compared with cache line size 128 bytes, and will introduce high rates of false sharing, degrade performance greatly. Therefore it is suggested that cache line size should be minimized.

Above half of local miss, 59.15% in rate and 64.45% in data percentage is served from remote cache, and in more detail, similar OS kernel cache miss is served from remote cache as User mode cache miss (remote KRN hit rate 62.70%, remote USR hit rate 41.33%, remote KRN hit data percentage 68.65%, remote USR hit data percentage 45.03%). It make sense since both OS and application are booted from one node, run on every nodes.

True sharing pattern of kernel mode and user mode memory access also share similar pattern. True sharing miss only accounts for a small percentage in remote cache hit (11.98% in rate, 7.96% in data volume) and local cache miss (7.09% in rate, 5.13% in data volume). Kernel true sharing accounts for large part of total true sharing miss, 77.80% in rate and 68.34% in data volume. User mode true sharing only accounts for 22.20% in rate and 31.66% in data volume.

Most of remote hit is false sharing, and it account for 88.01% in rate, 92.03% in data volume for remote hit, or 52.06% in rate, 59.31% in data volume for local miss. It is really a waste of performance since false sharing is avoidable in nature. Solutions should be taken to minimize the amount of false sharing. Detailed analysis shows that kernel mode accounts for most of false sharing, 89.01% in rate and 89.23% in data volume. Code false sharing miss is not distinguished from data false sharing miss. Considering that self-modifying code seldom happens, we can state that most of false sharing in kernel mode is caused by data cache miss.
False sharing misses can degrade performance greatly although they only accounts for a fraction of total memory access. Each false sharing miss will involve at least one network data communication, and some remote cache operation on behalf of it. How the data and control information is transferred between nodes is significantly important for performance. Suppose 1" level cache hit cause 1 cycle, 2" cause 5 cycles, local memory access cause 100 cycles, remote cache access time ranges from 50 cycles to 500 cycles. If we don’t take into the effect of L1 cache, then local hit stall would be 5 cycles. Total memory operation stall is like following figure. From that figure, it is suggested that high-end design is preferred to minimize the stall caused by remote cache hit. Grouping CPUs into clusters can significantly reduce the cost of data cache communication miss rate. And cache coherence mechanism in SMP can be utilized fully to minimize the remote cache operation caused by cache misses which cannot be served in local node.

Relevant research in CC-NUMA OS memory access pattern shows that data cache stall time is highly concentrated in a small number of routines. Small number of routines account for 96% of OS data cache stall time. Therefore binary translation technology can be taken to change the memory access pattern of those routines. For example, decrease the size of cache line to minimize the possibility of cache level false sharing.

![Graph showing remote cache hit stall and stall time](image)

- **Page Level Simulation Results**

Global physical memory is assumed under cache level to support memory access request. It is obvious that without page migration and replication technology, it is almost impossible to achieve considerable
good performance. Also sub-block technology is integrated in order to minimize the size of conflicting memory unit.

Therefore two configuration threshold should be selected, migration/replication threshold (if page miss number is more than this threshold, the page will be replicated or migrated to other node), and thrash threshold (if page has been replicated or migrated among nodes more often than this threshold, no more replication/migration will happen to this page, and sub-block technology will try to partition this block). The goal is that for any particular node, the percentage of local machine memory access is optimized as large as possible

as migration/replication threshold become larger, it is more difficult for a page to be migrated or replicated. Therefore, more page level memory access happen remotely, causing the protocol communication volume goes up, also the local hit access rate goes down. From figure, (128,32) degrade performance greatly compared with (32,32). Therefore, the migration/replication threshold should be a small number, to start replication/migration mechanism as early as possible.

Another phenomenon can be observed also, that the thrash threshold should be a small number also. Examining the (32,4), (32,8), (32,32) configuration, the higher the thrash threshold, the more difficult for a page to stop migration/replication, causing additional communication. The local hit access is also hurt since that kind of page causing noise in page scheduling. Hence the thrash threshold should also be a small number. In case of remote access of that page, sub-block mechanism is enabled to split that page, using the split fractions are page scheduling unit.
From figure it is observed that as the migration/replication threshold goes down, more page is split into sub-blocks. Not all sub-blocks are accessed or touched evenly when OS and application runs. It seems that high replication/migration threshold can effectively filter out “hot” page. When “hot” page is split into sub-block, the total touch on sub-block will goes up.

Another phenomenon observed from sub-block is that, almost 80-90% “hot” page to be split is accessed from kernel mode, that means, OS routines accounts for most of “hot” page migration/replication in system.

5. Conclusion and Future work
The memory and communication subsystem of virtual machines for cluster computing faces the same great challenges posed on other systems designs for issues of memory access locality, coherence and consistency, false sharing etc.

Running a single virtual multiprocessor across clusters seems problematic at this point due to many OS characteristics. Most commodity OSes are not aware of the locality and placement of a typical NUMA memory system. The paradigm of VMM providing the single uniform physical memory image from its distributed machine memories might be feasible for hardware-supported systems, but could face many serious problems if the network bandwidth and latency cannot compare to that of NUMA interconnection due to its heavy kernel level data sharing and locking activity. In fact, the locking and exclusiveness aspects of many kernel functions make OS the bottleneck for scalability. Our simulation results demonstrated that OS kernel pose severe problems under our web and database intensive benchmarks.

Compared with the uniform physical memory provided by VMM, user level shared virtual memory avoids the kernel part of memory communications at the cost of invoking the heavy OS kernel overhead for its normal communication, which could be about 2 orders expensive than the raw networking interface. More simulation and work needed to make sure which is better for what kind of workloads.

A high end design with specific hardware support may solve many of the problems, leading to a solution looks like a compromise between pure commodity SMP cluster and full custom-designed NUMA systems. It may cost less and provide similar performance with more flexibility. However, this needs more future work.

More detailed and exact simulations are needed for better evaluation of the ideas proposed here. Although we got some preliminary results. They are problematic in itself and not convincing for drawing any definite conclusions at this point.

We also only focused on memory and communication aspect of such kind of an interesting scheme. However, there are much more issues need further exploration. Like in most large scale machines, fault tolerance and containment is an important topic. As mention earlier in the report, there are many, many interesting topics on virtual machines for cluster computing.

6. Related Projects and References.
1. DISCO, Cellular DISCO, Hive Cellular OS and FLASH projects, Stanford.
2. Berkeley NOW project.
3. Princeton SHRIMP project.
4. TreadMarks project at Rice.
5. Wisconsin Wind Tunnel and Condor, Wisconsin.
6. DEC-WRL Shasta project.
7. NUMAchine, and Tornado/Hurricane projects in Toronto.
8. Shared Virtual Memory. Memory coherence by software. Kai Li, & P. Hudak
10. TreadMarks: Shared Memory Computing on Networks of Workstations. Cristiana Amza, Alan L. Cox, Sandhya Dwarkadas, etc.
12. Virtual memory mapped network interface for the SHRIMP Multicomputer. Matthias A. Blumrich, Kai Li, Richard Alpert, etc.
18. User-Space Communication: A Quantitative Study. Soichiro Araki, etc.