The following paper was originally published in the

Proceedings of the 3rd USENIX Windows NT Symposium

Seattle, Washington, USA, July 12–13, 1999

HIGH-END WORKSTATION COMPUTE FARMS USING WINDOWS NT

Srinivas Nimmagadda, Joshua LeVasseur, and Rumi Zahir

High-End Workstation Compute Farms Using Windows NT

Srinivas Nimmagadda, Joshua LeVasseur, Rumi Zahir
{ snimma@scdt.intel.com, jlevasse@mipos2.intel.com, rumi.zahir@intel.com }
Intel Corporation

Abstract

This paper describes our experiences in building and deploying Windows NT* based high-end workstation compute farms within the Intel engineering community. An overview of Intel's compute requirements is presented, along with the solution developed to address Intel's large compute cycle needs. The paper discusses NT's contribution to the solution, including recognized NT strengths as well as migration and deployment challenges we faced. The paper emphasizes workarounds to known issues for other user communities to leverage, but also enumerates areas in which Windows NT and the Win32 API* could be improved. Our paper concludes that despite many challenges, NT based high-end workstation farm computing is viable.

1 Engineering Computing Environment at Intel

As in many other semiconductor and computer systems companies, product development and engineering work for chip design, CAD tool development, and commercial EDA tool deployment has traditionally been carried out on Unix* based workstations. High-end workstation computing at Intel largely revolves around providing compute cycles to the various chip design projects for running a variety of computationally intensive workloads. Typical applications range from large compute-bound integer and floating-point applications (such as performance, logic and circuit simulations), to long-running jobs with very large data sets (e.g., full-chip layout verification or performance simulations of on-line transaction processing systems). Most of the jobs are compute-bound and usually run for several hours. It is not unusual for many of the compute-bound jobs to run for several days or even weeks. As a result, most of the compute cycles are spent by applications running in batch mode on a farm of hundreds of high-end multi-processor workstations. Usage models and system needs of a workstation compute farm are quite different from the more widespread file server, database client/server or web server installations typically found in large enterprises.

At Intel, hundreds of engineers submit thousands of compute and data intensive simulation jobs into the workstation compute farm every day. Jobs are typically a group of collaborating applications rather than a single executable. Within a job, applications are sequenced by scripts and communicate through files, named pipes or shared memory. Table 1 summarizes typical job parameters from an Intel workstation compute farm hosted on Windows NT.

This paper describes the challenges and experiences that we have faced in developing and deploying Windows NT based compute farms for Intel chip design computation needs.

Farm Characteristic	**Typical Environment**
# of Dual-processor Systems	> 100
# of Jobs per Week	> 25000
Utilization (user+system/wallclock)	> 80 %
Job Characteristics	
➢ # of Processes (per job)	1-10
➢ Memory requirements (per job)	50-500 MB
➢ Input Data Set Size (per job)	30-600 MB
➢ Job Runtime	Several minutes to several days

Table 1. Typical Workstation Farm Parameters

* Third party trademarks and brands are the property of their respective owners.

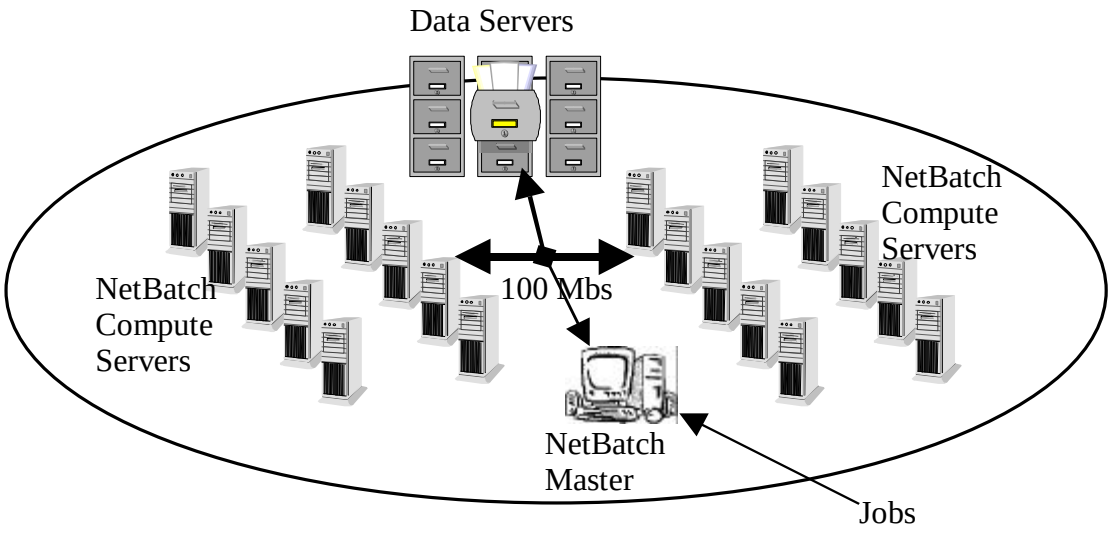

Figure 1. A NetBatch Compute Farm

2 Compute Farm Building Blocks

Our compute farms consist of hundreds of networked NT workstations built from off-the-shelf components. The machines are placed in rack and stack setting without monitor and keyboard/mouse attachments. Typical configurations include dual-processor Intel® Pentium® II 400 MHz CPUs, 512MB RAM, 9 GB local disk space, connected to data servers via 100Mb Ethernet. All machines run the Windows NT 4.0 Workstation operating system image with service pack three.

From the end user's point of view, the entire farm looks like a single virtual computer that provides on-demand compute cycles. The physical location of job execution is transparent to the user and results are reported as if the job executed locally, i.e. through output files. Typically, users develop and debug applications and scripts interactively using smaller data sets. Once debugged, the jobs run on the NT compute farm with much larger data sets, often through hundreds of iterations. Three key aspects of batch computing on Windows NT that we address in this paper are:

The Batch Computing Engine: The most important component in building a compute farm is a distributed job scheduling and execution engine. The engine provides a virtual computer view of the farm machines, transparent job execution, load balancing, and priority/quota based job-scheduling features. Section 3.1 presents NetBatch, an in-house batch engine, that we developed to address these requirements.

The Network File System: To provide batch jobs with the illusion of a single virtual computer, a unified run-time environment and name space are required at three different levels:

- a high-performance unified file system (data) view,
- a unified secure data access mechanism, and
- a unified user profile (path, environment variable and application settings).

Section 3.2 presents various file system and data access mechanisms used by our compute farm. These technologies include Samba* (public domain), DFS* (from Transarc) with DCE* (from Gradient), and NTFS*/SMB* servers (from Microsoft).

Application Development and Porting Issues: Most compute jobs consist of multiple collaborating applications that are glued together with perl or tcsh scripts. Porting of these legacy scripts to NT has often required more effort than actually porting the applications themselves. In cases where scripts are heavily dependent on idiosyncracies of Unix commands (such as ls, cat, grep, awk, sed, etc.), porting of legacy scripts just requires too much effort. Another paper in these proceedings [MTEX], describes a solution that allows us to run applications on NT while the bulk of the scripts execute on Unix. In Section 3.3 (in this paper), we discuss our experience using the public domain Cygnus utilities [Noer 98] on NT directly, and we identify key technical short-comings of NT's native scripting support.

3 Challenges and Solutions

3.1 NetBatch: A Distributed Batch Computing Engine on Windows NT

Figure 1 shows a high level view of a NetBatch compute farm. A NetBatch pool is a network of NT workstations, each running a NetBatch server process, which communicates with a pool master. The NetBatch Server process monitors the individual machine load and resources, and provides the seamless job execution environment. Each NetBatch pool has one pool master controller responsible for providing the user interface (for job acceptance, control, etc.), load-balancing and job scheduling capabilities. The load balancing service on the master collects load and machine availability information from the machines within the cluster, and selects least loaded machines for job execution. The scheduler service applies a set of criteria (based on priorities and user quotas) to the queued batch jobs to select the best job for execution.

The following sections highlight some of the key areas of our batch engine implementation:

- **Job Tracking and Management:** how NetBatch executes and tracks batch jobs with multiple child processes.
- **User Impersonation:** how NetBatch provides transparent execution of single or multi-process jobs under the exact user credentials and environment that existed at the time of submission.
- **Job Scheduling and Suspension**: how NetBatch load balances the compute farm by dispatching jobs to idle or least loaded machines, and permits suspension of running jobs.
- **Batch Computing on Interactive Desktop Workstations**: how NetBatch uses idle desktop workstation compute resources.

3.1.1 Job Tracking and Management

To carry out its functions, NetBatch must track batch jobs that spawn multiple child processes. For example, to track the resource usage of a batch job, and to suspend or terminate the job in response to a user request requires knowledge of a complete process tree of the job. The first approach we considered used a technique (similar to one implemented in the Microsoft SDK `tlist` command) to identify parent-child relationships by walking the kernel process data structures to construct the process tree. However, termination of an intermediate process splits the process tree and complicates group tracking. Another technique that we explored attaches a dummy object to the parent with a child inheritable handle. Then all processes containing a handle to the dummy object can be counted as part of the same process group. This method is also infeasible because applications can disable handle inheritance across process creation. We tried using session identifiers to provide another possible solution because each batch job impersonates the user (submitter of the job). This technique, however, does not work in scenarios where a job requires true process group support from the OS. Examples of this include application initiated process group control where a process can switch between groups and other accounting purposes. This prompted us to pursue a native NT solution with Microsoft, which has resulted in a new Windows 2000 feature known as job object [Solomon 98]. A job object is a new kernel object that provides capabilities such as process grouping, resource usage collection (on a group basis), and enforcement of process resource limits. Our conclusion is that job object feature is well suited for job tracking and management under Windows 2000, while any of the above workarounds provide reasonable solutions with restricted scope under NT 4.0 release.

3.1.2 User Impersonation

All processes in a batch job need to run with the exact user credentials of the person submitting the batch job. NT impersonates a user with the `LogonUser()` and `ImpersonateLoggedOnUser()` APIs. However, to use the `LogonUser()` API, the NetBatch Server requires the user's clear text password. There could be a significant time lapse between the NetBatch job submission and machine availability in the pool for execution of the job, during which the password may change.

To provide a clear text password to LogonUser(), NetBatch could collect the passwords at the time of job submission and securely store them until job execution. Alternately, all passwords could be collected in advance and stored in a two-way encrypted user password database for later retrieval to use with the `LogonUser()` API. However, maintaining a two-way encrypted database poses security concerns and also creates administration and password synchronization issues. This prompted us to investigate further and resulted in the creation of a password-less impersonation mechanism in NT. We enhanced the winlogon local security authority system with our representative DLL, which permits the NetBatch batch job to login without a password. However, this approach has a limitation that the impersonated process cannot access SMB network files.

In a Unix environment, user impersonation is a simple task with the `setuid()` system call, where the administrator (root) can impersonate a user for executing the batch job. Due to these reasons we feel that a true, trusted delegation-based impersonation capability would be extremely useful in NT.

3.1.3 Job Scheduling and Suspension

Load balancing between machines is an important factor in improving the overall throughput in the NT batch farm. Choosing the machine with the least-loaded CPU and the most suitable free memory, page file, and free temporary disk space configuration is an important function to optimize in a farm environment. In Windows NT, these indicators can be obtained from the system supplied performance counters. The NetBatch server periodically scans these counters and reports these machine availability indicators (CPU load, free memory, disk space, etc.,) to the NetBatch master for making best scheduling decisions. Since NT doesn't have a native performance counter that gives CPU load, we had used a formula similar to the one used to compute CPU load factor in Unix operating systems. This load factor is computed as the weighted moving average of the number of ready to run threads over 15 minutes. The formula for the load factor is $(10*N_1 + 4*N_5 + N_{10})/15$; where N_x is the average number of ready to run threads in the system during the last x minutes. It will be an interesting research topic to create a general-purpose machine availability indicator(s) based on available free CPU, memory, page file, and free disk space indicators, and further quantifying job run-time requirements using these indicator(s).

One of the features of our batch engine is to suspend execution of batch jobs on demand. Suspension of the job requires suspension of each process in the batch job. However, Windows NT doesn't provide a process suspension feature, although it can suspend individual threads in a process with the `SuspendThread()` `API`. However, this Win32 API function requires a handle to each thread in the process. Using the performance counters, it is easy to enumerate the thread IDs in each process. Win32 doesn't provide a mechanism to convert a thread ID to a thread handle! We can obtain a process handle given a process ID, but there is no way to suspend a process. This deficiency in NT is worked around using process debug APIs to obtain the thread handles. All threads are suspended resulting in process suspension. It would be very useful if the Win32 API supported a built-in process suspension mechanism, or exposed the internal APIs to convert a thread ID to a thread handle.

3.1.4 Batch Computing on Interactive Desktop Workstations

Intel deploys large numbers of powerful NT desktops for interactive desktop use. One of our goals is to utilize the unused computing bandwidth of these desktops during off-hours. The following are some key challenges we have encountered and addressed in providing farm computing using interactive desktops.

Detecting when a machine is being used interactively is an important component for integrating a user's desktop machine with the NT compute farm. NetBatch uses the Windows NT `SetWindowsHookEx()` API to setup hooks that monitor the keyboard and mouse activity and thus detect the presence of interactive users. After detecting a user, NetBatch takes appropriate intrusion avoidance actions including: preventing new batch jobs from beginning while the user is present, suspending already running jobs, and optionally terminating and resubmitting jobs to a different machine. While detecting the presence of console users is easy, detecting remotely logged-in users, such as logins from home using *telnet*, *rcmd*, *rsh* services, is a challenging task. We have identified a workaround for this by polling the process tree and identifying the unique users every few seconds. However, we found this type of polling is expensive when combined with the batch server's other tasks. We feel this is an important counter that the NT core OS should add to the list of performance counters. Without OS support, this is an expensive and sometimes inaccurate task to perform at user level.

Several of our batch jobs run for days to weeks. One of the capabilities we would like to have in the core OS is support for application checkpoint and migration from one machine to another. This feature would allow the batch engine to free up a desktop machine when the user resumes interactive work. At present, we have solved this problem using a combination of methods. The first approach temporarily suspends the batch job when the interactive desktop user is present and restarts the job when user leaves. The second approach suspends the job for a period of time, and if the interactive user is still present , terminates the job and restarts it on a different machine.

This impacts the overall throughput and turn-around time for the jobs. Another approach requires application-level state checkpointing and resumption . A group at Intel [Srouji 98] developed a transparent checkpointing method on top of NT, but it places limitations on the type of applications that can be checkpointed. Middleware approaches require

application source modification, or recompilation, or re-linking. This is not favourable in a typical EDA environment with integrated suites of applications from external and internal sources. Despite this, most middleware approaches also have limitations on check-pointing multi-process applications, or applications with IPC and other process dependencies. Our conclusion is that complete process checkpoint and migration is difficult without support from the core OS.

Disabling application popup error dialog boxes on a per process basis is very important. This is very useful especially if the batch jobs run on interactive desktops. A critical error in a batch program should not display a dialog box on the interactive desktop. Apart from interactive users not liking the popup display, it suspends any running batch jobs until someone hits OK or Cancel on the dialog box. Using the existing NT API `SetErrorMode()`, it is possible to disable pop-ups for batch applications.

3.1.5 Batch Engine Performance

Our NetBatch masters run on either Unix or NT platforms and manage NT farm servers. Our master code is mostly algorithmic (rather than system) code and the performance numbers for masters were roughly equivalent on both Unix and NT platforms. Master takes about 21% CPU for controlling and load balancing a cluster of 1000 machines. Average NetBatch overhead for managing a job through its life cycle (queuing, scheduling, providing execution environment and utilization metrics logging) is 400 milliseconds. The overhead on NT server is about 110 ms for starting the job, while 40ms on a comparable Unix machine. Of the additional 70ms overhead on NT, about 30ms were spent simulating fork() like API, 20ms were spent simulating the password-less (`setuid`) style code, and the rest for user environment translation and other NT specific overheads. This additional overhead is negligible, as it is a one time task required for launching a job that could run for long time. While the job is running, job monitoring (such as suspension, altering priorities, termination, etc.,) on NT has additional overheads due to lack of process groups and resource usage information. This was observed as additional 0.5-1% CPU consumed by our NT server during the job runtime for executing workarounds described in section 3.1.1 and section 3.1.3.

3.2 The Network File System

To make a user job run on any machine in the compute farm, we needed a uniform environment at three levels: a uniform file system space, a uniform data access control mechanism, and uniform user login profiles. Our batch compute jobs cover a wide variety of file access patterns: read and write to small data files, sequential read or write to multi-gigabyte data files, random access to large databases, repeated demand loads of executable pages, and reuse of the files. As a result, any data access mechanisms have to provide excellent network file system performance, and scale to facilitate concurrent data accesses to hundreds of batch farm clients.

Intel's support structure favors centralized file servers for easy administration and maintenance operations such as backup, retrieval, and project resource partitioning. Locating file data in central servers, and making the user login access profiles uniform across all the machines (interactive and batch) is the first step in providing a unified job run-time environment.

Along with NT farms, we have a large install base of Unix workstations that use NFS and AFS file systems. Applications in both NT and Unix environments need access to common data input files. To serve these cross-domain requirements we have considered three types of file or data sharing approaches: Samba, DFS, and native SMB (NT file servers).

Samba: Samba is an open source product that acts as a gateway between Unix file systems and SMB protocol based file clients. The Samba server runs on a Unix machine and can integrate file systems from different servers and export them to NT clients as a single share. It has the advantage of providing access to the unified name space of the Unix file system, but suffers from scalability and performance issues. We partially solved the performance problem by replicating the gateways and thus reducing the clients per gateway ratio. In our environment, logic simulation jobs required a ratio of one Samba gateway (running on a powerful Unix workstation) per 100 clients, where each client accessed 10-15MB of data over a period of a couple hours. The Samba solution did not satisfy the needs of architectural performance simulation jobs, which often have a large file working set (gigabytes of trace files), and demand load many pages from the executables.

DFS: DFS is a distributed file system from Transarc, which offers a global file name space shared by all clients, hides the disjoint nature of the DFS servers, offers local disk caching for improved performance, and provides ACL based file access controls. DFS file systems offer great flexibility and performance. In our environment we have chosen DFS for our data intensive compute farms. Initial versions of DFS lacked support for multi-user access on NT clients and experienced occasional stability issues. We worked with DFS/DCE

vendors (Transarc and Gradient) to add the multi-user features to DCE2.2 and DFS. This latest release also addressed several stability issues that existed in earlier releases of DCE and DFS.

SMB: We also tested the use of NT file servers with SMB based NT clients. The NT file servers provide access to their files through shares, which the NT client can access through UNC paths, or by mapping the share to a drive letter. UNC is not popular as it lacks support for standard file system operations such as changing or setting the current working directory. Microsoft Dfs offers an integrated name space solution for SMB clients, but it was dropped due to reliability problems with the earlier version. The drive letter approach was the only choice left for data access on NT clients but it also has a few issues. The drive letters are established globally throughout the OS, rather than on per user or session basis, which restricts the number of concurrent mappings, and opens security holes in a multi-user environment. NT also leaves stale mounts when a user login session terminates without unmapping a drive.

SMB suffers from reliability and scalability problems. Its reliability suffers from packet time-outs, which cause application level errors. Application binaries stored on a remote NT file server, when executed on a NetBatch compute farm machine, often experience abnormal program terminations due to packet time-outs when the OS attempts to demand load an unmapped code page. This problem can be solved by making a local copy of the image file, either by explicitly copying the file, or by setting a bit (/swaprun:net) in the image file which triggers the NT loader to make a local copy in the pagefile prior to execution. The main disadvantage of this approach is reduction in throughput and bottlenecks at the file server, especially when binary files are large and many of them execute concurrently on hundreds of farm machines. This problem has been addressed in other environments, such as on NFS based Unix clients, where the process execution is kept on hold state until normal communication with the file server is restablished. This mechanism isolates long running applications from transient network data access failures.

Filesystem Reliability Comparison: We attempted to compare the reliability of DFS, SMB with an NT server, and SMB with a Samba server exposing NFS file systems, within our compute farm environment. The workload was based on architectural performance analysis of gigabytes of compressed trace information from an online transaction processing benchmark. The workload is I/O bound, although a fair amount of CPU time is required for decompression. The data fetch and the data decompression are pipelineable, if the file system supports a streaming prefetch mechanism. The reliability experiments executed in the standard compute farms during weekends. The compute farms continuously handle a load under indeterminate network conditions. But restricting the data collection to weekend runs helped isolate the file servers from the spurious network traffic due to interactive users. The performance data was captured while executing one, two, five, and ten simultaneous jobs in a NetBatch cluster. All jobs requested the same file information from the server. Executables were located within the DFS filesystem, to protect the jobs from packet timeouts during demand loads of code pages. Table 2 tabulates the results. DFS successfully handled the load. SMB with the NT fileserver suffered from data packet timeouts under more intense loads. The Samba server and NFS servers were too loaded to return requests within the packet grace period; a single Samba server is the point of entry for all architecture related NT clients into the NFS file system.

SMB Summary: We suspect that poor caching by SMB causes an increased load on the network, which raises the probability of a packet time-out and thus a loss in reliability. SMB's reliability problems limit its scalability in our batch computing environment.

We abandoned the SMB/NT file server solution due to the above reasons, and due to the lack of robust cross-domain (NT to Unix client) capabilities. There are two key areas which NT must address to make the native SMB file system an acceptable solution in a typical engineering computing environment: i) improve reliability and scalability, and ii) improve client side

Filesystem	**1 job**	**2 jobs**	**5 jobs**	**10 jobs**
DFS	8.71 hours/job	8.0 hours/job	9.95 hours/job	10.85 hours/job
SMB with NT fileserver	8.45 hours/job	9.68 hours/job	failure	failure
SMB with Samba and NFS	failure	failure	failure	failure

Table 2. Filesystem reliability for a disk intensive workload (several gigabytes).

share mapping mechanisms (improved UNC and a superior approach to the multiple drive letters).

3.3 Application Development and Porting Issues

Many of our engineering applications were originally developed and deployed on Unix systems. While many solutions exist that emulate Unix services on NT such as OpenNT* [Walli 97], NutCracker*, and Cygwin* [Noer 98], we decided to convert our applications to NT's rich native Win32 API in most cases. For the large part this has been straightforward, especially since many of our applications are non-graphical in nature. In some instances, however, we have been unable to convert applications to the Win32 API, and in these cases, we use the Cygwin libraries and Cygnus utilities. We use these public domain tools for scripting, and for porting code that requires the use of the UNIX fork() system call, which is not provided by Win32 API. It seems to us that these two capabilities would be good candidates for future Windows NT or Win32 API extensions.

3.3.1 Scripting Solutions

As stated earlier, many of our batch jobs string applications together using scripts. As a result, our design automation team chose to use Cygnus utilities and public domain perl/tcsh packages to support our batch mode scripting needs. While we have encountered several stability issues with the cygwin.dll b.17 version (related to concurrency and multi-user usage of the GNU commands), migration to cygwin.dll b.20 seems to have addressed the majority of our issues. Currently it looks like we will continue to use these public domain scripting engines, since the native Windows NT scripting command shell (cmd.exe) only provides very limited capabilities. For instance, it is missing regular expression parsing and string processing capabilities, and it also lacks an interface with the Win32 API. To provide a batch friendly environment, Windows NT needs a more powerful, customizable and rich native shell environment. Another fundamental shortcoming for scripting on NT is that the Windows NT executable loader only recognizes files with certain extensions as executables. This problem can be solved if the NT loader is modified to recognize scripts using the file execute attribute , and then use the first line of the script to determine which scripting engine to invoke.

3.3.2 Fork System Call

In the traditional Unix environment, the fork() system call has been conveniently used to replicate the process state and address space. Our simulators protect the state of long-running simulation jobs by periodically creating a child image. The parent image waits for completion of the child image. If the child image encounters live-locks or run-time errors, the parent simulator will terminate the child image and fast-forward the simulation beyond the condition causing the problem. The isolated address space of the child simulator insulates the parent from any of the child's errors and memory leaks. The child also has access to another 2 gigabyte address space for data collection. While the Windows NT kernel does provide process address space and some state cloning capabilities, the Win32 API stops short of providing a complete process clone interface, even for non-graphical "console" applications. The NT thread paradigm does not replace the benefits of the fork paradigm, for a faulty thread can damage the entire simulator. The one-for-one cloning of a process address space, along with associated operating system state semantics, is a powerful and much needed Win32 API.

4 Recognized NT Strengths

With the exception of earlier workarounds, NT provides many other attractive features, which we used to develop a robust and simplified NetBatch engine, some of which are listed here. The NetBatch components are highly multithreaded, using native kernel threads to handle intrusion detection, job control, resource monitoring, server communication, etc. Various components utilize asynchronous procedure calls (APC) and mail slots to create a robust communication infrastructure. Performance counters provide a central place to get a wide range of machine resource indicators and process metrics used for optimal job scheduling decisions. Process priority classes provide a way to utilize machine cycles with minimal intrusion with other tasks and improve latency for (with real time priority) critical tasks. SCM (Service Control Manager) provides a good way for installing and managing long running NetBatch services on hundreds of farm machines. The NetBatch engine utilizes the event log for debug and error analysis. Apart from these core OS facilities, Visual Studio (MSVC) Integrated Development Environment (IDE) and Win32 SDK provides excellent software development, debugging, and maintenance environment.

* Third party trademarks and brands are the property of their respective owners.

5 Related Work

Condor [Bricker 91, Litzkow 88] is a high-throughput batch-computing environment for networked workstations. It is sensitive to the needs of desktop users and makes use of unused desktop cycles. It uses flexible match making algorithms [Raman 98] to find the best machine to execute each job. Condor also provides transparent checkpointing and migration of batch jobs on intrusion detection. Some limitations of Condor include the unavailability of a complete batch solution on NT, and limitations on check pointing most multi-process, communication, and I/O intensive applications in the typical high-end computing environment. The distributed nature of Condors *schedd* makes enforcing user or group scheduling policies and fair share allocation difficult. The gateway architecture for multi-clustering makes implementation and managing cross-site policies e asier, but results in poor scalability and reliability due to the complexity of the remote scheduling protocols.

LSF (Load Sharing Facility) from Platform Computing Inc., is based on the Utopia architecture [Zhou 92]. The LSF suite has a complete set of job scheduling and load-balancing features. An NT port of LSF is also available. Some limitations of LSF on NT are the lack of transparent authentication and data access support, and lack of good cross domain (Unix-NT) seamless computing support.

Microsoft's Wolfpack [MSCS 98] provides clustering extensions to the Windows NT operating system that promise scalability and availability of servers in a mission critical enterprise environment. It has limited load-balancing features, but does not provide job-scheduling capabilities. Wolfpack can be used to scale and load-balance data servers, while the distributed batch engine described in this paper is able to schedule and load-balance jobs across NT workstation compute farms.

6 Summary

Intel's engineering community generates many compute intensive jobs, suitable for batch execution, which may sustain execution times of hours to weeks. Users see the batch pool as a large virtual computer that provides on-demand compute cycles while seamlessly executing the user jobs. Three key components necessary to establish the homogenous virtual computer is a batch compute engine, a unified job run-time environment, and application infrastructure to manage large workloads. The batch compute engine manages workstation resources, distributes load across many compute nodes, controls the jobs, and manages the job run-time environment. The unified job run-time environment ensures consistency between the interactive and batch compute environments, provides a global file name space, enforces a global security mechanism, and propagates application configurations.

As this paper describes, NT offers solutions (either direct capabilities or workarounds) for most of the large compute farm issues. At Intel, we successfully use NT compute farms for some of our mission critical needs. Our experience has shown that the overall NT environment stability is comparable to that of the Unix environment. Our conclusion is that NT is a viable choice as an OS to deploy for the infrastructure of a large compute farm, and offers tremendous cost/performance advantages to RISC/Unix workstation solutions. In this paper we have presented several challenges and solutions that other NT user communities can leverage, and presented areas in which Windows NT and Win32 API could be improved further.

The following lists offer some suggestions for improving Windows NT and the Win32 API.

Operating System Related Improvements:

- Provisions for secure user impersonation without using clear text passwords.
- Addition of a fork() system call.
- Improved shell support for scripting.
- Operating system support for process suspension, check pointing and restart.
- Support for a true multi-user execution environment in Windows NT.

File System Related Improvements:

- Improve network file system reliability and scalability.
- Support for a unified name space and better client side share access features.

Future improvements to our batch engine described in this paper include better intrusion detection for local and remotely logged-in users, improved scheduling schemes, and better support for multi-cluster capabilities.

Acknowledgements

Thanks to Tae Paik, Eldon Chan and other members of the corporate NT engineering computing program for their great encouragement and constant support for this effort. Thanks to Raghu Krishnamurthy, Ty Tang, and Mike Hester for their contribution in developing the

batch engine on NT during early 97 and Dave Liebson for his contributions in addressing several farm issues. Thanks to Drew Hess and Tom Willis for providing valuable technical feedback on this paper. The progress we have made couldn't have been possible with out the help from many other members of our IT NT team and great co-development effort from our ever demanding internal customers.

References

[Bricker 91] Allan Bricker, Micahel Litzkow, and Miron Livny: "Condor Technical Summary", University of Wisconsin – Madison, 1991.

[Litzkow 88] M.J. Litzkow, M. Livny, and M.W. Mutka. "Condor – A Hunter of Idle Workstations". In Proc 8th International Conference on Distributed Computing Systems, San Jose, California, June 1988.

[MTEX] T. Tang, V. Lal, and S. Krishnapura, "MTEX: A Bridge For Migrating CAD Design Environment From UNIX To NT", Usenix Windows NT Symposium, 1999.

[MSCS 98] White paper on "Windows NT Load Balancing Service (Wolfpack) technical overview", *http://www.microsoft.com/ntserver/ntserverenterprise/techdetails/prodarch/wlbs.asp* and *clustarchit.asp.*

[Noer 98] Geoffrey Noer, "Cygwin32: A Free Win32 Porting Layer for UNIX Applications", Proceedings of 2nd USENIX Windows NT Symposium, Seattle, Washington, August 3-4, 1998.

[Raman 98] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed Resource Mana gement for High Throughput Computing, *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, July 28-31, 1998, Chicago, IL.

[Solomon 98] David Solomon, "Inside Windows NT", Second Edition, Microsoft Press.

[Srouji 98] Johny Srouji, Paul Schuster, Maury Bach, and Yulik Kuzmin: "A Transparent Checkpoint Facility on NT", Proceedings of 2nd USENIX Windows NT Symposium, Seattle, Washington, August 3-4, 1998.

[Walli 97] Stephen Walli: "OPENNT™: UNIX Application Portability to Windows NT™ via an Alternative Environment Subsystem", Proceedings of USENIX Windows NT Workshop 1997, Seattle, Washington, August 11-13, 1997.

[Zhou 92] S. Zhou. LSF: Load sharing in large-scale heterogeneous distributed systems. In Proc. of Workshop on Cluster Computing, 1992.

[Zhou 92] S. Zhou, J. Wang, X. Zheng, and P. Delisle. Utopia: A Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems, Technical Report CSRI-257, University of Toronto, Toronto, Canada.