# MOSIX
# parallele Verarbeitung mit transparenter Prozessmigration

Tobias Sandhaas

21. November 2002

# Inhaltsverzeichnis

# Kapitel 1

# Einleitung

## 1.1 Was ist MOSIX

MOSIX ist ein Akronym für "**M**ulticomputer **O**perating **S**ystem for Un**ix**" und stellt ein Forschungs- und Entwicklungsprojekt für ein dezentrales Rechensystem dar. Es wird seit 1981 an der Universität von Jerusalem entwickelt und seit jeher von Professor Amnon Barak betreut. Erste Verwendung fand es in einem Clustersystem von PDP-11 Computern auf Basis des Betriebssystems UNIX Version 7, und wurde im Lauf der Jahre stetig auf aktuellere Unix Versionen und Hardware portiert. MOSIX war in den ersten Jahren hauptsächlich eine Sammlung von Algorithmen die immer wieder an gesponsorter Hardware getestet worden ist. Mitunter förderten Ministerien verschiedener Länder beispielsweise die U.S. Airforce und das israelischen Verteidigungsministerium das Projekt. MOSIX fand aber erst in den frühen 90er Jahren Anwendung ausserhalb der Universität von Jerusalem, nachdem es für das Betriebssystem BSD unter Intel-PCs portiert worden ist. Die aktuell neuste Version 1.5.2 von MOSIX steht mittlerweile unter der GNU Public License und basiert auf dem aktuellen Linux Kernel 2.4.13 für Intel-PCs.

## 1.2 Ziele von MOSIX

Unter MOSIX ist die Benutzung der Workstations und Server innerhalb des Cluster unkompliziert, sodass es für den Anwender wie ein Einzelplatzrechner erscheint. Die Anwenderprogramme müssen nicht modifiziert oder auch mit speziellen Bibliotheken verlinkt werden damit sie im Cluster verteilt ablaufen. Die Kernel-basierten adaptiven Ressource-Sharing Algorithmen sind ge-

trimmt auf hohe Performanz, ausserordentliche Skalierbarkeit ohne zusätzlichen Overhead und eine sehr transparente Benutzung. Sie sind so entworfen, dass sie je nach Lastverteilung im Cluster Prozesse vorbeugend und auch während der Laufzeit mehrfach migrieren können. Zudem wird ebenfalls der Speicherverbrauch im Cluster kontrolliert um Seitenfehler zu reduzieren und die Geschwindigkeit zu steigern.
Kapitel 2 beschreibt diese Eigenschaften näher.

## 1.3 Typische Anwendungsgebiete

MOSIX eignet sich besonders gut für wissenschaftliche Berechnungen, wie beispielsweise dem "Traveling Salesman"-Problem. Diese sind in der Regel sehr rechenlastig und lassen sich zudem leicht in mehrere Prozesse aufspalten. Hat man Anwendungen, die sehr viele Prozesse erzeugen, kann man zudem eine Software wie PVM [1] verwenden, welche es ermöglicht Prozesse fern zu erzeugen und somit zu einer besseren initialen Lastverteilung führt. Prinzipiell ist MOSIX nicht auf bestimmte Anwendungsfälle zugeschnitten, sondern universell einsetzbar und liefert hierbei gute Ergebnisse.
Aufgrund der ausserordentlichen Transparenz eignet sich MOSIX nahezu für jeden Rechenpool und führt zu einer besseren Ressourcenverwendung dieser. Da die Verteilung adaptiv ist, gibt es kaum Performanzeinbußen, falls sich ein Nutzer lokal auf einen Knoten des Clusters einlogt und selbst rechenintensive Programme (zum Beispiel netscape) startet. MOSIX benötigt zudem keine einheitliche Rechnerkonfiguation innerhalb des Clusters und könnte somit auch für viele gemischte Rechnernetze im Einsatz sein.

[1] Parallel Virtual Machine - http://www.epm.ornl.gov/pvm/

# Kapitel 2

# Systembeschreibung

## 2.1 Die Hauptcharakteristika von MOSIX

Im Folgenden werden die Hauptcharakteristika von MOSIX detailiert beschrieben.

- Netzwerktransparenz
  Für den Anwender wirkt das MOSIX-Cluster wie ein Einzelplatzrechner. Seine Applikationen laufen in der für ihn gewohnten Art und Weise ab und müssen von ihm nicht besonders behandelt werden. Die Ressourcen des Clusters werden "automatisch" genutzt und können zu einem schnelleren Programmablauf führen. Es ist hierbei nicht erforderlich, dass der Anwender den realen Standort des migrierten Prozesses oder seiner Daten kennt, da dies transparent auf Kernelebene vom MOSIX System ausgetauscht wird. Wenn der Prozess beispielsweise ein Systemaufruf tätigt, ist es die Aufgabe spezieller Kernelmodule den Datenaustausch mit anderen Prozessoren im Cluster durchzuführen. All dies findet ohne Zutun des Anwenders statt und wird daher nicht von ihm wahrgenommen. Man könnte daher den MOSIX-Cluster wie eine große "Virtuelle Maschine" ansehn.

- Autonomie der Rechner
  Das MOSIX System setzt die Eigenständigkeit der einzelnen Rechner voraus. Die Algorithmen unterscheiden die Prozessoren des Clusters nur nach ihrer Last und verteilen die Prozesse dementsprechend. Deshalb ist es wichtig, dass die Rechner voll funktionsfähig sind und somit jegliche zugewiesene Aufgabe lösen können. Jeder Rechner muss jeden anderen Rechner innerhalb des Clusters ersetzen können. Hier-

zu benötigt jeder Rechner eine ähnliche Softwareausstattung und der MOSIX Kernel muss geladen sein. Die Hardware kann nahezu frei gewählt werden, lediglich der CPU-Befehlssatz muss übereinstimmen. Das Betriebssystem braucht selbstverständlich einen uneingeschränkten Zugriff auf die Hardware um alle gestellten Aufgaben bewältigen zu können. Die Autonomie ist für viele Fähigkeiten des MOSIX Systems von grundlegender Bedeutung wie man beispielsweise an der "Dynamischen Konfiguration" erkennt.

- Zufallsgestützte Informationsverbreitung
  Um eine optimale Lastverteilung im Cluster zu bewirken, benötigt man Informationen anderer Prozessoren. Diese Informationen beinhalten die CPU-Last, die CPU-Geschwindigkeit, den freien Speicherplatz und die einzigartige Prozessornummer und wird in festen Zeitintervallen an eine zufällig ausgewählte Menge von Rechnern des Clusters verschickt. Jeder Rechner führt eine eigene kleine Tabelle dieser empfangenen Informationen und bereinigt diese regelmässig von "zu alten" Einträgen. Durch diese Methodik erhält jeder Prozessor auf sehr kostengünstige Art Informationen, die für die Lastverteilung nötig sind. Man spart sich dadurch das aktive Nachfragen und erhält zudem eine elegante Möglichkeit das System dynamisch zu verändern. Jeder Prozessor ist somit auf einfache Art und Weise steuerbar; man braucht lediglich seine Informationen nicht mehr weiterzureichen um ihn aus der Prozessmigration zu entfernen. Das Gleiche gilt auch für Erweiterungen. Hierbei hat dieser Ansatz einen weiteren Vorteil, da ein neuer Prozessor im Verbund nicht aufgrund der initial geringen Last von allen anderen Prozessoren zunächst als Migrationshost verwendet wird. Das System ist aufgrund der zufallsgestützten Informationsverbreitung zudem noch fehlertoleranter, da ein defekter Rechner innerhalb des Clusters automatisch nicht als Migrationshost ausgewählt werden kann.

- Dezentralisierte Administration
  Der MOSIX-Cluster wird dezentralisiert gemanaged. Aufgrund seiner Autonomie hat jeder Rechner lokal absolute Entscheidungsrechte und ist unmittelbar zuständig für alle lokalen Objekte. Es existieren daher keine Master-Slave Beziehungen unter den Rechnern des Clusters sowie allgemein zuständige Server Rechner. Lediglich die lose Kooperation der Rechner wird vorausgesetzt. Hierbei wird auch keine Bedingung an die Netzwerktopologie gestellt, lediglich eine beliebige Verbindung muss zum Zwecke des Datenaustauschs existieren. Die dezentralisierte Administration ermöglicht eine dynamische Konfiguration des Clust-

ers mit nur minimalen Unterbrechungen auf Kosten einer leichteren Administrierbarkeit.

- Erhöhte Ausfallsicherheit
  Da jeder Rechner jeden Anderen im Cluster ersetzen kann und die Administration dezentral erfolgt, bieten MOSIX-Cluster auch eine erhöhte Ausfallsicherheit. Jeder migrierte Prozess wird mittels Zeitmesser vom Home-Node [1] kontrolliert und gegebenfalls, falls der Prozess sich nicht mehr meldet, erneut auf einen anderen Prozessor migriert. Der defekte Rechner wird dann nach einiger Zeit aufgrund der zufallsgestützten Informationsverbreitung auch nicht mehr für die Migration herangezogen und fällt automatisch aus dem Clusterbetrieb. Sollte jedoch der Home-Node ausfallen ist der gesamte Prozess unwiderruflich verloren, weil dann der "Grundstock" des Prozesses wegbricht, welcher beispielsweise für die Kontrolle und Datenerfassung der migrierten Prozesse zuständig ist. Werden im Cluster auch dedizierte Dateiserver verwendet, die beispielsweise mittels "Fibre Channel" mit dem Cluster verbunden sind, so steigert sich auch hier die Redundanz.

- Dynamische Konfiguration
  Die dynamische Konfiguration erlaubt jedem Rechner den Cluster ohne besondere systemweite Neukonfigurationen zu verlassen oder beizutreten. Einfache Einstellungen an dem jeweiligen Rechner genügen meistens. Zudem besteht die Möglichkeit den Cluster in Gruppen (Subcluster) zu unterteilen und den Grad der Kommunikation dieser dann festzulegen. Eine Gruppe kann dann zum Beispiel einem bestimmten Projekt zugewiesen werden. Bei der intergruppen Kommunikation kann man zwischen "vollem Zugriff", "teilweisen Zugriff" und "keinem Zugriff"' wählen. Der "volle Zugriff" entspricht hierbei dem normalen Clusterbetrieb und der "teilweise Zugriff" beinhaltet nur den Zugriff auf entfernte Daten ohne Prozessmigration.

- Skalierbarkeit
  Der MOSIX-Cluster ist aufgrund der dynamischen Konfiguration leicht erweiterbar und so entwickelt, dass er auch bei größeren Clustern bestmögliche Performanz bietet. Idealerweise sollte die Performanz des Clusters linear mit der Anzahl der Prozessoren steigen. Um den tatsächlichen Performanzgewinn bei MOSIX darzustellen hat das Entwicklerteam einige Testergebnisse veröffentlicht. Beispielsweise wurde mit Hilfe von MOSIX die Bewegung von Molekülen simuliert, was

[1] Home-Node - Der Rechner an dem der Prozess initial erzeugt worden ist

ein sehr rechenintensives Problem darstellt. Die *Abbildung 2.1* zeigt die Leistungssteigerung des Clusters in Abhängigkeit der Prozessoren, wobei in diesem Fall aufgrund des relativ kleinen Problems schon bei 16 Prozessoren kaum noch Verbesserungen auftreten. Man erkennt, dass die Lastverteilung des MOSIX Systems bei 8 Prozessoren zu einer 6.5fach schnelleren Berechnung führt, was nahezu dem Optimum von 8 entspricht.

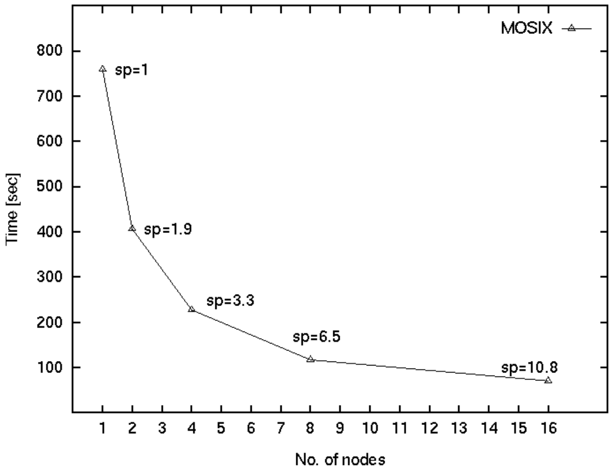


**Abbildung 2.1:** Leistungssteigerung bei Skalierung

Damit MOSIX möglichst skalierbar bleibt wurde stets vermieden Services zu entwerfen, deren Last mit der Prozessorzahl steigt. Denn diese könnten spätestens ab einer gewissen Systemgröße ihren Aufgaben nicht mehr nachgehen und würden letztendlich bis zu einem totalen Stillstand des Clusters führen. Dies ist auch einer der Gründe, die zur dezentralen Administration führten. Aufgrund weiterer Überlegungen wurden folgende Prinzipien festgehalten:

1. *Konstante Servicenutzung*: Die Anfrage an einen Service im Cluster muss konstant sein und darf insbesondere nicht von der Prozessorzahl abhängen.
2. *Symmetrie*: Jeder Rechner muss völlig eigenständig und unabhängig von anderen seine Aufgaben bewältigen können.
3. *Client-/Server Protokoll*: Jede Kommunikation soll ein Mindestmaß an Rechnern benötigen, vorzugsweise zwei.
4. *Teilinformationen*: Jede Entscheidung soll aus einer begrenzten Menge von Informationen des Clusters getroffen werden.

5. *Willkürlichkeit*: Die Informationen des Clusters (z.B.: Lastverteilung) soll willkürlich an bestimmte Rechner verschickt werden.

- Kompatibilität
  MOSIX war viele Jahre "AT&T UNIX System V." kompatibel und setzt seit einigen Jahren auf Linux auf. Hierbei wurden die bestehenden Kernelschnittstellen nicht verändert, sodass das System nach wie vor kompatibel zu vorhandener Linux Software ist. Selbst Software wie das zuvor erwähnte PVM, welche es ermöglicht Prozesse fern zu erzeugen und daher selbst gewissen Modifikationen am Kernel vornimmt, kann mit wenig Aufwand integriert werden.

- Kernel-Kommunikation
  Es wurden spezielle Erweiterungen am Kernel vorgenommen um die Kommunikation zwischen migrierten Prozessen und dem Home-Node effizienter zu gestalten. So realisierte man mitunter Kernelfunktionen, welche Systemaufrufe auf anderen Rechnern tätigt oder auch Daten unter Rechnern austauscht. Die Kommunikation im Netz ist durch ein eigens entwickeltes Protokoll realisiert, welches geringe Anfangslatenz und hohen Durchsatz aufweist und direkt mit den entsprechenden Kernelfunktionen kommunizieren kann. Das Protokoll basiert auf TCP/IP und setzt ein verlässliches Transportmedium voraus.

- Hauptspeicherverwaltung
  Schon bei der Migration wird überprüft, ob der vermeintlich freie Speicherbereich des ausgewählten Rechners überhaupt ausreicht den Prozess mit seiner Umgebung aufzunehmen. Sollte nun zu einem beliebigen Zeitpunkt der freie Speicherbereich eines Rechners einen Schwellwert unterschreiten, werden selektiv Prozesse auch auf Kosten der Lastverteilung wegmigriert um Seitenfehler zu reduzieren. Als Resultat wird der global verfügbare Speicher des Clusters besser von den Prozessen ausgelastet, sodass seltener ausgelagert werden muss. Dadurch wird der relativ langsame Vorgang der Ein- und Auslagerung von Speicherseiten reduziert und trägt in der Regel auch zu einem Performanzgewinn bei.

- Dynamische Lastverteilung durch Prozess Migration
  Die Grundlage einer funktionierenden Lastverteilung im MOSIX-Cluster ist vorallem die zufallsgestützte Informationsverteilung. Der Algorithmus, der für die Lastverteilung zuständig ist, wertet die erhaltenen sowie die eigenen Daten über die Prozesse aus und versucht die Gesamtsituation durch Migration zu verbessern. Hierbei können

eigene als auch Gastprozesse migiert werden, wobei ein einzelner Prozess möglichst nicht auf mehr als 2 Rechner aufgeteilt wird. Der völlig selbständige Algorithmus teilt zudem dem Home-Node mit, wo dessen Prozess nun läuft damit dieser nicht durch die Migration verloren geht. Die Hauptspeicherverwaltung kann jedoch die Vorgaben des Algorithmus der dynamischen Lastverteilung überstimmen und zu einer eher untypischen Lastverteilung im Cluster führen. Das Ziel des Algorithmus ist eine gleichmässige Lastverteilung unter allen Prozessoren des Clusters zu jedem Zeitpunkt der Ausführung.

## 2.2 Prozessmigration

### 2.2.1 Steuerung

Die Entscheidungsfindung zur Migration eines Prozesses ist die wichtigste Komponente des MOSIX Systems. Hierzu verarbeitet das System Eigenschaften der Prozesse, die in der Prozessstruktur des Kernels abgelegt sind, sowie Informationen über die Auslastung anderer Prozessoren im Cluster. Ziel ist eine möglichst homogene Verteilung der Auslastung innerhalb des Clusters. Die Informationen in der Prozessstruktur des Kernels beinhalten beispielsweise die verbrauchte Rechenzeit, die Anzahl der erzeugten Kinderprozesse als auch die getätigte Ein- Ausgabe(I/O) einzelner Prozesse mit den jeweiligen Rechnern des Clusters und veralten mit einer einstellbaren Rate.

Bei SMP-Systemen werden die Prozessoren zwar getrennt betrachtet, trotzdem wird berücksichtigt, dass eine Migration innerhalb dieses Systems weniger "kostet" als zu anderen Rechnern, da der Kommunikationsaufwand, vorallem bei Systemaufrufen, wesentlich geringer ausfällt.

Diese Steuerung beachtet auch, dass ein zuvor vom Hauptspeicherverwaltungsalgorithmus getätigte Migration nicht erneut auf eben diesen Rechner migriert wird um die Migrationskosten durch Falschauswahl gering zu halten.

Zudem wird stets der lokale Prozessor bei der Entscheidungsfindung bevorzugt und schneller eingestuft als er tatsächlich ist. Dadurch wird verhindert, dass zwei ähnlich ausgelastete Prozessoren, die sonst keine Aufgaben übernehmen, ständig einen Prozess hin- und hermigrieren.

Ausserdem muss die bis dato verbrauchte Rechenzeit des Prozesses einen gewissen Wert überschritten haben, man spricht dann von "residency", um ihn migrieren zu dürfen. So wird verhindert, dass Prozesse ständig migriert werden und letztendlich nie Rechenzeit zugewiesen bekommen.

### 2.2.2 Deputy-Remote Mechanismus

MOSIX wurde entwickelt, um eine völlig transparente Migration von Prozessen zu ermöglichen. Hierzu wird nachdem der Entschluss zur Migration feststeht, der lokal gestartete Prozess des Home-Nodes in den User-Kontext und System-Kontext geteilt. Der System-Kontext ist hierbei rechnerabhängig und beinhaltet alle prozesseigenen Daten und Strukturen im Kernelspeicher wie beispielsweise den Kernelstack und offene Deskriptoren. Er wird *Deputy* genannt und verweilt zu jedem Zeitpunkt der Ausführung auf seinem Home-Node. Der User-Kontext wird *Remote* genannt und beinhaltet hauptsächlich die Prozessdaten im Benutzerspeicher und sonstige zur Ausführung relevante Daten. Zu denen gehört mitunter der Programmcode, der Stack, die Prozessregister und die Rohdaten des Prozesses ansich. Dieser Teil ist rechnerunabhängig und kann im Lauf der Anwendung auch mehrfach den Prozessor wechseln. Der Austausch von Kommandos und Daten zwischen dem *Deputy* und *Remote* wird über den sogenanten "Link-Layer" gesteuert und über eine dedizierte TCP/IP Verbindung mittels eines speziell entworfenen Protokolls realisiert.

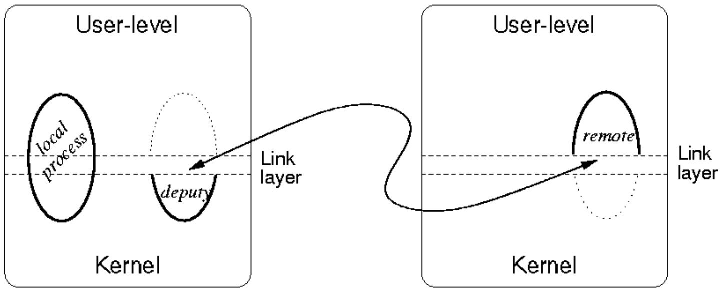


**Abbildung 2.2:** Aufbau eines lokalen und migrierten Prozesses

Da die Kommunikation zwischen Kernel- und Benutzerbereich unter Linux durch Systemaufrufe[2] und spezielle Funktionen (zum Beispiel: copy_to_user()) eindeutig definiert ist, fasst der "Link-Layer" jegliche Systemaufrufe ab und untersucht, ob eine netzwerkübergreifende Kommunikation nötig ist. Es wird hierbei versucht möglichst viel lokal auf dem

[2] engl.: system calls

Gastrechner auszuführen um die performanzreduzierende Kommunikation im Netzwerk einzuschränken. Die Weiterleitung von Systemaufrufen erfolgt durch den "Link-Layer" völlig transparent, sodass der Benutzer in der Regel davon nichts merkt.
MOSIX unterscheidet bei Anweisungen zwischen synchronen und asynchronen Anweisungen. Der Programmablauf wird bei synchronen Anweisungen wie sie die Systemaufrufe darstellen wie gewohnt unterbrochen und bei erhalt der Rückgabe fortgesetzt. Für asynchrone Anweisungen wie beispielsweise den Signalen kontrolliert der *Remote* bei jedem Kontextwechsel im Prozessor den Kommunikationskanal und leitet gegebenfalls daraus resultierende Aktionen ein.

Ein paar typische Situationen bei der Prozessmigration werden im Folgenden erläutert:

- Signale und Wake-Ups[3]
  Diese werden üblicherweise vom Kernel des Home-Nodes an den *Deputy* übergeben und von diesem überprüft, ob gewisse Aktionen ausgeführt werden müssen. Der Kommunikationskanal wird hierbei für die Weitergabe dieses asynchronen Ereignisses verwendet. Der Remote auf dem Gastrechner leert gegebenfalls direkt nach dem Kontextwechsel im Prozessor noch vor der Fortsetzung des eigentlichen Programms diesen und führt die nötigen Operationen auf dem Gastrechner aus.

- Netzwerk Sockets
  Netzwerk Sockets, welche für die Kommunikation im Netzwerk zuständig sind, können derzeit nur im Home-Node erstellt werden. Deshalb werden migrierte Prozesse die viel I/O benötigen zu einer erhöhten Netzwerklast führen und damit zu einer Reduktion der Gesamtperformanz des Clusters beitragen. Der "Link-Layer" übernimmt hierbei wieder die Aufgabe der Weiterleitung von Daten und Anweisungen.

- Datei-Deskriptoren
  Bei der Migration von Prozessen spielen bereits geöffnete Datei-Deskriptoren eine wichtige Rolle. Diese können aufgrund des festgelegten Programmablaufs nicht einfach geschlossen und auf dem Gastrechner erneut geöffnet werden und verhalten sich daher analog zu den "Netzwerk Sockets". Der "Link-Layer" fängt alle Zugriffe des *Remote* auf diese Deskriptoren ab und leitet diese zum *Deputy* weiter.

[3]Wake-Ups können beispielsweise mit select() überprüft werden

Datei-Deskriptoren werden ansonsten lokal auf dem Gastrechner geöffnet um eben diese Performanzeinbußn aufgrund der Netzwerkkommunikation zu reduzieren. Das DFSA[4] stellt hierbei die Schnittstelle zwischen dem Prozess und den benötigten Datei-Deskriptoren dar und bietet mitunter eine Zugriffsmöglichkeit auf alle Dateien im Netzwerk an.

- Speicherseiten
  Jeder Prozess hat Seitenverwaltungstabellen in denen mitunter der Status und der Ort der Prozessseiten[5] des Prozesses festgehalten sind. Unter MOSIX sind "dirty-pages", das sind Seiten die modifiziert wurden und gleichzeitig noch im physikalischem Speicher stehen, von besonderer Bedeutung, da diese initial bei der Migration von Prozessen mittransferiert werden. Die restlichen dementsprechend kürzlich weniger benützten Seiten werden erst bei Bedarf kopiert um die Last im Netzwerk zu reduzieren und die Programmausführung zu beschleunigen. Man spricht hier von "demand-paging". Damit diese "fernen" Seiten bei den natürlich auftretenden Seitenfehlern[6] auch gefunden werden, werden die Seitenverwaltungstabellen des Gastprozesses nach dem Kopieren angepasst. Dementsprechend finden sich mit der Zeit auf dem *Deputy* nur noch nicht benutzte Seiten[7], wogegen der Rest auf dem Gastrechner aktiv Verwendung findet. Um die ohnehin schon enormen Einbußen, die durch das Kopieren über das Netzwerk herbeigerufen werden zu reduzieren, verwendet MOSIX verschiedene Buffer und Pre-Caching Mechanismen auf Kernelebene.

### 2.2.3 Migrationszeit in Abhängigkeit der Prozessgröße

Die Prozessmigration wird in zwei Schritten ausgeführt. Zunächst wird die Kommunikation der zwei Rechner hergestellt und das Prozessgerüst auf dem Gastrechner erstellt. Im zweiten Schritt werden dann die Prozessseiten über das Netz kopiert und im Gastrechner neu erzeugt. Während der erste Schritt der Prozessmigration fixe Kosten hat, verursacht jede geladene Seite im zweiten Schritt linear zusätzliche Last und kann die Ausführung verzögern. Um nicht stets die gesamte Prozessgröße zu kopieren, werden initial nur die "dirty-pages" und später bei Bedarf die restlichen Seiten nachgeladen.

[4] Direct File System Access
[5] engl.: pages
[6] engl.: page-fault
[7] engl.: clean pages

Die *Abbildung 2.3* zeigt die Migrationskosten in Abhängigkeit der übertragenen Prozessgröße. Für diese Testreihe dienten Pentium-Pro 200Mhz Rechner, welche mit einem 100MBit-Ethernet (78.8 Mbit/s) oder Myrinet Lan (228.8 MBit/s) verbunden waren. Die Migrationsverzögerungen durch den ersten Schritt der Migration sind respektive drei bzw. zwei Millisekunden.

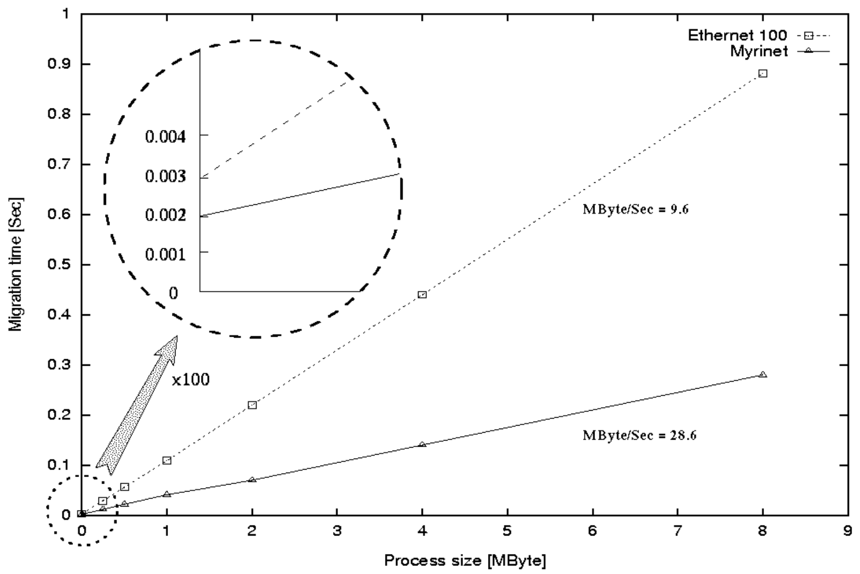


**Abbildung 2.3:** Migrationszeit in Abhängigkeit der Prozessgröße

## 2.3 Dateisystem

### 2.3.1 Direct File System Access

Das DFSA ist der Bestandteil des MOSIX Systems, welche alle Zugriffe auf Datei-Deskriptoren steuert. Hierbei wird versucht möglichst viele Dateizugriffe lokal zu halten um die Performanzeinbußen gering zu halten und nur die wirklich nötigen Zugriffe wie beispielsweise auf schon geöffnete Dateien des Home-Nodes über das Netzwerk zu leiten. Aufgrund der häufigen Dateizugriffe innerhalb der Rechner des Clusters hat man zudem einen Algorithmus entworfen, der überprüft, ob und wie viel I/O ein Prozess mit verschiedenen Rechnern tätigt und legt dem Prozess gegebenfalls eine Migration zu einem Rechner mit dem er besonders viel kommuniziert nahe.

An das DFSA sind gewisse Bedingungen geknüpft, die das verwendete Dateisystem voll unterstützen muss:

- Konsistenz der Dateien
  Sobald ein Prozess eine Datei auf einem beliebigen Rechner des Clusters verändert, müssen alle anderen Rechner diese Änderung sofort oder bei dem nächsten Zugriff eines Prozesses sehen. Dateisysteme wie das "Global File System" (GFS) lösen das Konsistenz Problem durch ausgefeilte Lösungen bei denen Hardwareressourcen gleichzeitig verschiedenen Rechnern angeboten werden.
- Konsistenz der Zeitstempel
  Dateien innerhalb eines Rechners müssen unabhängig von dem Rechner aus dem Modifikationen herrühren konsistente und fortschreitende Zeitstempel aufweisen.
- vereinheitlichter Mountpoint für alle Dateien im Netzwerk
  Alle Langzeit-Speichermedien des Clusters müssen vereinheitlicht über einen Mountpoint (beispielsweise /mnt/rechner-nr/) erreichbar sein um einen generischen Zugriff auf systemweite Dateien zu ermöglichen. Dies schliesst den jeweiligen lokalen Rechner ein.

Das DFSA trägt zu einem großen Teil dazu bei den Overhead bei der Kommunikation innerhalb des Clusters zu reduzieren und spielt daher für die Performanz eine wichtige Rolle.

### 2.3.2 MOSIX File System

Das erste Dateisystem unter MOSIX, welches DFSA einsetzt und dessen Anforderungen realisiert, ist das "MOSIX File System" (MFS). Es implementiert den einheitlichen Mountpoint zur Addressierung aller Dateien im Cluster, indem man anstatt der sonst üblichen obersten Wurzel / einen neuen Wurzelknoten angibt */../* , zum Beispiel */mnt/*. Beispielsweise kann man nun systemweit die Datei */etc/passwd* des Rechner mit der Nummer 2 mit */mnt/2/etc/passwd* ansprechen. Alle Dateizugriffe, ob nun lokal oder auf andere Rechner, verwenden für alle Zugriffe diese generische Methode, was dieses System auch sehr flexibel und gut skalierbar hält. Die Konsistenz der Daten und Zeitstempel wurde dadurch gelöst, indem man nur einen Cache direkt auf dem "Dateiserver" zulässt. Dabei werden die Operationen auf den Dateien nur von dem betreffendem Dateiserver ausgeführt und bleiben daher zu jedem Zeitpunkt konsistent. Wird beispielsweise eine Datei über Rechnergrenzen hinweg geöffnet, so wird diese Operation völlig transparent zu dem Dateiserver geleitet, dort lokal geöffnet und steht nun dem Clientrechner über einen Datenkanal zur Verfügung. Aufgrund dieses Datenflusses zwischen den Rechnern ist dieses Verfahren nicht sehr performant, aber die Konsistent ist zumindest zu jedem Zeitpunkt der Ausführung gewährleistet.

### 2.3.3 Global File System

Das GFS ist ein kommerzielles Produkt der "Sistina Software" und speziell für den Einsatz als Dateisystem unter Linux Clustern entwickelt worden. Es lässt sich ohne weitere Modifikationen unter MOSIX-Clustern einsetzen und kann mittlerweile MOSIX´s DFSA verwenden. Durch das GFS lassen sich Speichergeräte[8] von verschiedenen Rechnern zeitgleich gemeinsam nutzen und auch im Betrieb administrieren. Da kein Rechner Steuerungsaufgaben erledigen muss und die Datenverbindung zwischen den Rechnern und diesen Speichergeräten durch dedizierte Leitungen vorzugsweise "Fibre Channel" sichergestellt wird, ist das System fehlertoleranter. Die einzelnen Rechner des GFS sehen das Dateisystem der Speichergeräte lokal und können daher wie üblich darauf zugreifen. Die Bedingungen des DFSA wie beispielsweise die Konsistenz der Dateien und Zeitstempeln sowie ein vereinheitlicher Mountpoint werden im GFS vollständig erfüllt.

### 2.3.4 Vergleichstest verschiedener Zugriffsmethoden

Um die Leistungssteigung des Clusters unter Verwendung des DFSA aufzuzeigen, muss das Dateisystem des Clusters stark belastet werden wie es beispielsweise bei einem Mailserver auftreten kann. Der sogenante "Postmark"-Test [9] erzeugt einige Dateien verschiedener Größe und wendet einige Transaktionen (mit Unteraktionen) darauf an. Dies wird solange wiederholt bis eine gewisse Last erreicht wird und dann die Ausführungsdauer für 10 Transaktionen gemessen.
In unserem Szenario prüfte man ein MOSIX-Cluster von identischen Pentium 550Mhz Rechnern auf denen Linux mit der Version 2.2.16 läuft. Das Ergebniss von 4 verschiedenen Methoden des Dateizugriffs mit variierender Blockgröße ist in folgender Tabelle einsehbar.

[8] engl.: storage devices
[9] Postmark Benchmark - $http://www.netapp.com/tech_library/3022.html$

| | Blockgröße des Datentransfers | | | | | | |
|---|---|---|---|---|---|---|---|
| Zugriffsmethode | 64b | 512b | 1Kb | 2Kb | 4Kb | 8Kb | 16Kb |
| Lokale Ausführung | 102.6 | 102.1 | 100.0 | 102.2 | 100.2 | 100.2 | 101.0 |
| MFS mit DFSA | 104.8 | 104.0 | 103.9 | 104.1 | 104.9 | 105.5 | 104.4 |
| NFS | 183.3 | 169.1 | 158.0 | 161.3 | 156.0 | 159.5 | 157.5 |
| MFS ohne DFSA | 1711.0 | 382.1 | 277.2 | 202.9 | 153.3 | 136.1 | 124.5 |

Leider gibt es keine Referenzwerte für das GFS.

Die erste Zeile listet das Ergebnis auf, wenn die Ausführung nur lokal getätigt wird. Bei "MFS ohne DFSA" werden Mechanismen die fest an das DFSA gekoppelt sind deaktiviert, dazu gehören zum Beispiel die Migration eines Prozesses bei hoher I/O Last und die Verwendung des lokalen Dateisystems wenn es die Situation zulässt. Aus der Tabelle ergibt sich eine durchschnittliche Verschlechterung der Ausführung von 3.7% zwischen "MFS mit DFSA" und der "lokalen Ausführung". Im Vergleich zu "NFS" schneidet "MFS mit DFSA" rund 59% besser ab. Zudem ist MFS bei Blockgrößen über 4Kb performanter als NFS und daher eine geeignete Alternative bei Dateisystemen in Clustern. Der Performanzeinbruch bei kleinen Blockgrößen wird hervorgerufen durch den initialen Verbindungsaufbau, da der Clientrechner unter MFS keinen Dateicache pflegt.

## 2.4 Geplante Erweiterungen des Systems

### 2.4.1 Network Ram - Distributed Shared Memory

MOSIX's derzeitige Hauptspeicherverwaltung migriert einen Prozess der viele Seitenfehler produziert auf einen Rechner mit ausreichend freiem Speicher aus. Dies funktioniert nur, wenn auch wirklich ein Rechner die Seiten des Prozesses in seinen Hauptspeicher aufnehmen kann. Um dieses Problem zu umgehen wird derzeit an dem *Network Ram* gearbeitet, der es ermöglichen soll die Prozessseiten auf mehreren Rechnern zu verteilen, damit eine möglichst optimale Nutzung der Speicherressourcen im Cluster mit möglichst wenig Seitenfehlern gewährleistet ist. Zudem wird der Einsatz von "Distributed Shared Memory"-Lösungen auf Basis des *Network Ram* untersucht, um den Overhead bei kommunikationslastigen Prozessen zu reduzieren. Das Ziel der neuen Mechanismen ist den Prozess analog zu DFSA möglichst optimal zu den gegebenen Daten im Cluster zu migrieren, wobei diese Daten auch auf mehreren Rechnern verteilt sein können.

### 2.4.2 Migrierbare Sockets

Ein Problem kommunikationslastiger Prozesse unter MOSIX ist, dass die Netzwerksockets stets nur auf dem Home-Node des Prozesses erzeugt werden können. Dies hat zur Folge, dass bei einer Kommunikation zwischen zwei migrierten Prozessen die Daten bis zu drei mal über das Netz geschickt werden; zunächst von dem migrierten Prozess zu dessen Home-Node, von diesem dann zu dem Home-Node des Kommunikationspartners und von dort zu dessen migriertem Prozess. Zudem stellt bei dem derzeitigen Verfahren der Home-Node ein Bottleneck des Systems dar. Eine Lösung wären *migrierbare Sockets*, die eine direkte Kommunikation unter den migrierten Prozessen erlaubt und somit den Overhead in der Kommunikation reduziert.

# Kapitel 3

# Zusammenfassung

Insgesamt betrachtet ist MOSIX ein hoch performantes, sehr gut skalierbares Clustersystem, das sich aus einfachsten Bauteilen aufbauen lässt. Durch seine Eigenschaften ermöglicht es eine nahezu optimale Lösung vielfältiger Anwendungsszenarien und ist daher nahezu universell einsetzbar. Selbst bei undeterministischer und daher unbekannter Ressourcenverwendung der Prozesse kann MOSIX mittels seiner adaptiven Lastverteilung auch während der Laufzeit beste Ergebnisse erzielen. MOSIX zeichnet sich zudem durch seine sehr gute Ausfallsicherheit aus, die in der Autonomie der Rechner und dezentraler Administration begründet liegt. Die sehr gute Skalierbarkeit des Systems ist zudem ein weiterer Vorteil, die zu einem sinnvollen Einsatz von MOSIX in vielen Bereich führt.

# Literaturverzeichnis

Barak A., Guday S., Wheeler R. G.: The MOSIX Distributed Operating System, Springer Verlag 1993

Amar L., Barak A., Eizenberg A., Shiloh A.: The MOSIX Scalable Cluster File Systems for LINUX , July 2000

Barak A., Braverman A.: Memory Ushering in a Scalable Computing Cluster, Journal of Microprocessors and Microsystems, Vol. 22, No. 3-4, Aug. 1998

Barak A., La'adan O.: The MOSIX Multicomputer Operating System for High Performance Cluster Computing, Journal of Future Generation Computer Systems, Vol. 13, No. 4-5, pp. 361-372, March 1998

Barak A., Braverman A., Gilderman I., La'adan O.: Performance of PVM with the MOSIX Preemptive Process Migration, Proc. 7th Israeli Conf. on Computer Systems and Software Engineering, Herzliya, pp. 38-45, June 1996

McClure S., Wheeler R.: MOSIX - How Linux Clusters Solve Real World Problems, Proc. 2000 USENIX Annual Tech. Conf., pp. 49-56, San Diego, CA., June 2000

Internetpräsenz von MOSIX: http://www.mosix.cs.huji.ac.il/