MOSIX

Cluster-Computing für LINUX

und DFSA-File-I/O

MOSIX ist ein Software-Tool und unterstützt

- Cluster-Computing (CC)
- Adaptive Resource-Sharing-Algorithmen auf Kernel-Ebene
- Optimiert hinsichtlich:
  - High-Performance
  - Overheadfreie Skalierbarkeit
  - Einfache Verwendung

Zentrales Anliegen

- Kooperativer Betrieb eines Clusters aus Workstations und Servern
- Aus Anwendersicht ein SSI (“Single System Image”).

MOSIX ist flexibel:

- Reagiert auf veränderliche Resourcen-Nutzung im Cluster
- Transparente (präemptive) Prozess-Migration im Cluster
- Lastausgleichend, regierend auf Memory-Enpässe in einem Node
- Gesamtperformance des Clusters steht im Vordergrund
- Aus Anwendersicht gleichermaßen geeignet für sequenzielle und parallele Verarbeitung

Einstieg in MOSIX

Zentrales Anliegen: kooperatives Zusammenarbeiten mehrerer Nodes, so als ob es sich um **ein** System handelt.

Nodes können sein: **Workstations, Server (auch SMP/NUMA-Server).**

Vorteil von einzelnen SMP-Systemen:

- Gleichzeitiger Ablauf mehrerer Prozesse
- Automatische und sofortige Anpassung an veränderte Lastprofile
- Keine Anpassung durch den User erforderlich

Cluster - heterogene Hardware-und Software-Landschaft:

- Unterschiedliche CPU-Geschwindigkeiten
- Unterschiedliche Memory-Kapazität
- Unterschiedliche Technologien (SMP oder nicht-SMP)
- User verantwortlich für Zuweisung von Prozessen zu Nodes
- Auch bei homogener Systemlandschaft eingeschränkte Kooperationsfähigkeit wegen Festanbindung der Systemdienste an einen Knoten.

Software für Cluster-Computing:

- PVM
- MPI ...
- LSF auf Batch-Ebene
- Stark eingeschränkte Anpassungsfähigkeit an veränderliche Resourcen wie
  - Memory
  - I/O-Kapazität
  - Interprozess-Kommunikation
- Keine Absprache der User untereinander

Was ist MOSIX (www.mosix.org)?

- Tool für der UNIX-Familie, z.B. LINUX
- Bietet Algorithmen für Resource-Sharing
- Reagiert unverzüglich auf veränderte Resourcen-Belegung
- Verbesserung der Clusterweiten Performance, sowohl für sequenzielle als auch parallelisierte Aufgaben
- Stellt clusterweite Resourcen jedem Node zur Verfügung

Was ist MOSIX (www.mosix.org)?

- Bringt CC-Plattformen in die Nähe von SMP-Systemen
- Globale Resourcen-Zuteilung
- Lastverteilung über den Cluster
- Entpflichtet den User vom clusterweiten Management der Systemresourcen
- Kein Zwang für den User
- Unabhängig von der physikalischen Netzwerk-Technologie

MOSIX-Technologie: Mosix besteht aus zwei Teilen:

- Präemptive Prozess-Migration (PPM)
- Algorithmen für adaptives Resourcen-Sharing

Die Realisierung erfolgt Kernel-Ebene, aber unter Zuhilfenahme ladbarer Moduln, d.h. letztendlich bleibt der eigentliche Kernel unverändert.

Anwenderseitig bleiben die Systemschnittstellen unverändert.

Präemptive Prozess-Migration

Erlaubt den Transfer eines Prozesses jederzeit zu beliebigen Cluster-Nodes, basierend auf den Resource-Sharing-Algorithmen (Lastverteilung).

Eine Migration ist ebenfalls möglich, entweder synchron durch den Prozess selber, oder durch einen User-Eingriff von außen. Dies ist insbesondere nützlich beim Austesten von Scheduling-Algorithmen.

Der Superuser hat zusätzliche Privilegien, z.B. Änderung von allgemeinen Policies, oder Festlegungen über die Zugehörigkeit eines Nodes zum CC.

Im Sinne von MOSIX besitzt jeder Prozess einen **Home Node (UHN)**: Node, auf dem der Prozess, z.B. beim Login, kreiert wurde, oder z.B. bei PVM der Node, auf dem der PVM-Dämon die Prozess-Aufspaltung vorgenommen hat.

Das System-Image-Modell bei MOSIX ist ein CC mit folgenden Eigenschaften

- jeder Prozess läuft verwaltungsmäßig auf dem UHN
- jeder Prozess einer MOSIX-Session hat (verändernden) Zugriff die Umgebung.

Zu anderen Nodes migrierte Prozesse verwenden wenn möglich die lokalen Resourcen. Der Zugriff auf die Prozessumgebung erfolgt aber über den UHN. D.h. insbesondere:

- Ein ps-Kommando eines migrierten Prozessen zeigt alle (auch migrierten) Prozesse an
- Ein Zugriff auf die Uhrzeit mittels **`gettimeofday()`** liefert die Uhrzeit des UHN.

Vorgehensweise von PPM

Falls die Gesamt-Resourcenanforderungen eines Nodes unterhalb einer gewissen Schwelle bleiben: Prozesse bleibt an den UHN fixiert.

Bei Überschreitung von Schwellwerten werden geeignete Prozesse zu anderen geeigneten Nodes migriert mit dem Ziel der optimalen Auslastung der netzweiten Resourcen.

Kleinste migrierbare Einheit: der Prozess (keine Thread-Migration).

Die Migration eines Prozesses kann auch mehrfach erfolgen, falls später anderswo mehr Resourcen verfügbar werden.

MOSIX kennt keine zentrale Kontrollinstanz und keine Master-Slave-Organisation.

Jeder Node kann unabhängig von den anderen operieren und seine Entscheidungen treffen, d.h. optimaler Einsatz sowohl auf kleinen als auch großen CCs.

Skalierbarkeit wird erreicht durch:

- Einbindung des Zufallsprinzips in die MOSIX-Algorithmen
- Nur teilweise Kenntnis des Zustands anderer Nodes
- Regelmäßige Verteilung lokaler Zustandsinformation an andere Nodes
- Nodes entscheiden nur anhand der im jüngsten Zeitfenster empfangenen Zustandsinformation anderer Nodes

Resource-Sharing-Algorithmen

Beruhen auf Lastausgleich und optimierter Cluster-weiter Memory-Bewirtschaftung.

Der Lastausgleich versucht, dezentralisiert **Lastunterschiede** zwischen Paaren von Nodes **durch Prozessmigation** auszugleichen. Als Auswahlkriterien dienen beispielsweise:

- Anzahl der Prozessoren in einem Node
- Eigenschaften der Prozessoren eines Nodes (Geschwindigkeit,...)

Die Memory-Bewirtschaftung hat als Ziel, den clusterweiten Memory möglichst optimal auszunutzen, d.h ein Swapping der Prozesse zu verhindern.

Der Algorithmus wird aktiviert bei zu starkem Paging auf einem Node und hat stärkeres Gewicht als der Lastausgleich-Algorithmus, d.h. auch zu einem CPU-belasteten Node ist eine Migration möglich, falls damit das gesamte Paging reduziert wird.

Prozess-Migration

MOSIX unterstützt vollständig transparente Prozessmigration (PPM). Nach der Migration agiert der Prozess weiterhin mit seiner Umgebung, unabhängig vom Node. Dies wird ermöglicht durch eine Zweiteilung der migrierbaren Prozesse in:

- Den migrierbaren User-Kontext ("Remote")
- Den UHN-abhängigen, nicht migrierbaren System-Kontext ("Deputy")

Der Remote enthält den Programm-Code, Stack, Datensegmente, Memory-Tabellen und Register des Prozesses.

Der Deputy enthält eine Beschreibung der mit dem Prozess verbundenen Resourcen und einen Kernel-Stack für die Ausführung des Prozesses im privilegierten Modus.

Als Site-abhängiger Kontext-Träger muss der Deputy auf dem UHN fixiert sein, er wird also nie migriert. Das Interface zwischen Remote und Deputy unterliegt einem festen Protokoll, das so angelegt ist, dass es netztransparent ist (Link-Layer, spezieller Kommunikations-Kanal, TCP/IP).

Zeitlicher Aufwand für die Prozessmigation:

- Fester Anteil für die remote-Prozesserstellung
- Variabler linearer Anteil abhängig von der Zahl zu migrierender Seiten

Transparenz des migrierten Prozesses durch

- Forwarding Site-abhängiger System-Calls zum Deputy (UHN), synchrone Ausführung als Interaktion zwischen den beiden auf verschiedenen Nodes angesiedelten Prozess-Kontexten. System-Call wird vom Deputy ausgeführt, Ergebnisse an den Remote zurücktransferiert und Fortsetzung des Remote.

- Site-unabhängige System-Calls werden lokal vom Remote ausgeführt.

Zusätzliche, anders gestaltete Interaktion zwischen Deputy und Remote bei

- Signalzustellung
- Benachrichtigung nach Eintreffen von Events, z.B. eingelesenen Daten.

Notwendige Aktionen:

- Information des Deputy durch den UHN-Kernel
- Deputy überprüft, ob der Remote zu kontaktieren ist
- Remote-Benachrichtigung über einen speziellen Kommunikations-Kanal, der vom Remote ständig abgehört wird
- Remote-Reaktion mit Weitergabe an den User-Level

Nachteile:

- Overhead bei nicht lokal ausführbaren System-Calls wie z.B. bei Filezugriffen oder Socket-Operationen durch den Remote.
- In Planung: Migrierbare Sockets
- Workaround: Im-voraus-Prozess-Verteilung durch PVM oder MPI auf verschiedene Nodes.

Implementierung

Der Deputy als UHN-Stellvertreter braucht UHN-seitig keine User-spezifischen Seitentabellen zu verwalten.

Andererseits bedeutet ein Transfer von Information von/zum User eine Kommunikation zwischen Deputy und Remote, mit allen Nachteilen wie

- Protokoll-Stack
- Netzwerk-Latency

Lösung

Implementierung eines speziellen Caches Deputy-seitig, z.B. um beim erstmaligen Lesen aus Dateien durch Prefetching Daten auf Vorrat einzulesen.

Für memory-mapped Files auf Seiten des Remote hält der Deputy eine spezielle Tabelle.

Obwohl die User-Register normalerweise sich in Verantwortung des Remote befinden, können sie bisweilen exklusiv vom Deputy manipuliert werden.

Einschränkungen

Aus Sicherheitsgründen sind die Remotes für weitere auf dem Remotenode laufenden lokalen Prozesse nicht sichtbar, und auch umgekehrt nicht.

Eine Interaktion oder Manipulation durch lokale Prozesse ist nicht möglich. Lediglich eine weitere Migration durch den lokalen Systemverwalter ist zugelassen.

Spezielles Prozessverhalten wie

- Direkte Manipulation von I/O-Devices des UHN
- Zugriff auf den Shared Memory des UHN
- Teilnahme am Real-Time-Scheduling des UHN

verhindert entweder eine Migration oder bewirkt, wenn der Prozess bereits migriert ist, eine Rück-Migration zum UHN.

Informationsbeschaffung

Regelmäßiges Aufsammeln von Information über das Prozessverhalten bei

- Systemcalls
- Prozess-Zugriff auf User-Daten

als Entscheidungshilfen pro/contra Migration. Diese Statistik-Daten verfallen mit der Zeit, damit stets das aktuelle Prozessverhalten im Vordergrund steht. Die Statistik-Daten verfallen unverzüglich bei *execve()*-Systemcalls, da dann jegliche Prozessinformation obsolet ist.

Der Prozess selbst hat ebenfalls einen Einfluss auf Invalidierung der Statistik-Daten, wenn er z.B. Kenntnis über sein zukünftiges Verhalten hat.

MOSIX-API

Abgeändert durch standardisierten Zugang über das /proc-Filesystem, neuer Eintrag in /proc durch /proc/mosix; vermittelt Information wie:

- Synchrone und asynchrone Migrations-Requests
- Sperren eines Prozesses gegen automatische Migration
- Information über den momentanen Node
- Einschränkungen bei der Migration
- Setup und Administration
- Sammlung und Invalidierung von statistischen Daten
- Information über vorhandene Resourcen auf den konfigurierten Nodes
- Information über die Remote-Prozesse

Unterverzeichnisse:

- /proc/mosix/admin Administration
- /proc/mosix/info Clusterweite Information
- /proc/mosix/nodes/nnnnn Per-Node-Information
- /proc/mosix/remote/pppp Remote-Prozessinformation

Weiterer Zugriff auf Information

- Kommandos:
  - **`mon`**: Anzeige von Load, Geschwindigkeit, Ausnutzung und Speicherbelegung über den Cluster hinweg, greift auf /proc/mosix/info zu.
  - **`Mps`**
  - **`mtop`**: Clusterweite Ausprägungen von ps und top
  - **`mosctl`**: Node-Administration

- Java-Applet oder CGI-basierter Monitoring-Tools mit Informationsmöglichkeit über das Internet hinweg.

Änderungen am Kernel 2.2.x

- Etwa 80 neue Files (40.000 LOC)
- 109 modifizierte Dateien (7.000 LOC)
- 3.000 LOC für Load Balancing Algorithmus

Mosix für Kernel 2.4.x vor der Fertigstellung.

Geplante Erweiterungen

- Migrierbare Sockets zur Reduktion des Overheads bei Interprozess- Kommuniktion
- Migrierbare Tempfiles zur Kreation temporärer Dateien auf dem Remote-Node
- Neue Algorithmen zum besseren Resource-Management für
  - CPU
  - Memory
  - IPC
  - I/O

- Network-RAM für große Prozesse mit über den Cluster verteilen Memory
- (TreadMarks!), ganz anderer Ansatz.
- Erweiterung auf andere Plattformen: DEC Alpha, SUN Sparc,...

DFSA (Direct File System Access)

- Re-Routing-Mechanismen zur Reduktion des Overheads I/O-lastiger Systemcalls bei migrierten Prozessen
- Bekanntes Problem bei verteilten Dateisystemen.
- **Ziel**: lokale Ausführung der Systemcalls im aktuellen Node.
- Voraussetzung: Optimierung der Verteil-Algorithmen mittel- bis hoch-I/O-lastiger Prozesse
- Berücksichtigung von Nodes mit viel I/O.
- Bewirkt größere Flexibilität bei der Remote-Node-Auswahl.
- **MOSIX-Philosophie**: Prozess zu ihren Daten bringen, d.h. zum Node, für den das Filesystem lokal ist.

Problematisch

- Simultaner Zugriff des Prozesses zu Dateien auf verschiedenen Dateisystemen.
- Das Verhältnis zwischen CPU und I/O ändert sich zeitlich nicht vorhersehbar.
- Mehrere Prozesse zeigen ein solches Verhalten.

Arbeitsweise von DFSA

- DFSA entscheidet, wo File-I/O eines migrierten Prozesses auszuführen ist:
  - auf dem momentanen Node
  - oder dem UHN
- Verwendete Filesysteme sind clusterweit identisch gemountet, mit gleichen Mount-Flags
- Links sind clusterweit identisch
- Das UID/GID-Schema ist clusterweit identisch oder zumindest gefeit gegen unerlaubte Zugriffe

- Voraussetzungen erfüllt? File-Systemcalls sind auf dem momentanen Node, nicht auf dem UHN auszuführen
- Entspricht einem Redirect
- Durch DFSA permanente Synchronisation des Datei-Status (open/close/Dateizeiger) zwischen migriertem Prozess und UHN

DFSA setzt ein Filesystem voraus, für das gilt:

- Änderungen an einer Datei auf einem Node werden unverzüglich auf allen Cluster-Nodes sichtbar, d.h. z.B. Cache-Invalidierung bei einem Client-Server-Modell
- Dazu eignen sich am besten Filesysteme mit Shared-Hardware (SAN!)
- Konsistente Verwaltung der Time-Stamps für jeden Node
- Auf einem Node gelöschte Files/Directories werden erst dann freigegeben, falls von keinem weiteren Node Zugriffe offen sind

Geeignete Kandidaten sind z.B.

- Global File System (GFS)
- xFS
- FrangiPani

Das MOSIX File-System MFS

- MOSIX bietet mit **MFS** ein eigenes DFSA-fähiges Prototype-Filesystem
- **MFS** garantiert clusterweit eine einheitliche Sicht der Dateien und Directores aller gemounteten Filesysteme beliebigen Typs
- Files und Directories sind clusterweit zugänglich für alle Nodes

- Durch den MFS-Mountpoint **/mfs** bezieht sich **/mfs/1456/tmp/m** auf die Datei **/tmp/m** im Node **1456**.
- Folge: Skalierbarkeit, da jeder Node Client und Server sein kann.
- MFS garantiert Cache-Konsistenz, es ist nur ein Cache vorhanden
- Standard-LINUX-Platten- und Directory-Caches sind nur auf dem Server aktiviert, nicht aber auf den Clients.

Performance

Die folgende Tabelle gibt einen Ausschnitt aus einer Messung (Zeitangaben in Sekunden) wieder. Plattform: Kernel 2.2.16, Clustergröße: 2 x Pentium 550 Mhz mit IDE-Platten.

Blocksize [Byte]	Local im Server	MFS mit DFSA	NFS	MFS ohne DFSA
64	102.6	104.8	184.3	1711.0
512	101.1	104.0	169.1	382.1
1K	100.0	103.9	158.0	277.2
2K	102.2	104.1	161.3	202.9
4K	100.2	104.9	156.0	153.3
8K	100.2	105.5	159.5	136.1
16K	101.0	104.4	157.5	124.5