دانشکده برق و کامپیوتر
دانشگاه صنعتی اصفهان

openMosix

ابزاری برای توسعه کلاسترهای مبتنی بر لینوکس

گزارش سمینار درس سیستم عامل پیشرفته

مدرس:

جناب آقای دکتر داورپناه جزی

ارائه دهندگان:

نرجس خاتون حبیبی

زینب زالی

زمستان ۸۵

فهرست مطالب

۱- مقدمه ۳
۲- تعريف كلاستر ۳
۲-۱- تعريف كلاستركامپيوتري ۳
۲-۲- تقسيم بندي كلاسترهاي كامپوتري ۳
۳- معرفي openMosix ٤
٤- نصب openMosix ٥
٤-۱- كامپايل كردن كرنل لينوكس ٥
٤-۲- نصب و تنظيم ابزارها ٦
٥- تكنولوژي openMosix ۹
٥-۱- مكانيزم مهاجرت پروسه(PPM) ۱۰
٥-۲- الگوريتم هاي اشتراك منابع ۱۱
٥-۲-۱- الگوريتم موازنه بار ۱۱
٥-۲-۲- الگوريتم راهنماي حافظه ۱۲
٥-۳- مهاجرت پروسه(PPM) ۱۲
٥-٤- نكاتي در رابطه با مكانيزم remote/deputy ۱۳
٥-٥- سيستم فايل openMosix ۱٤
٥-٦- APIي openMosix ۱٤
٦- معرفي openMosixview ۱٥
٦-۱- openMosixview ۱٥
٦-۲- openMosixprocs ۱۸
٦-۳- openMosixcollector ۲۱
٦-٥- openMosixmigmon ۲٤
۷- جمع بندي ۲٤
منابع ۲٥

۱- مقدمه

تکنولوژی کلاسترینگ امکان می‌دهد که چند سیستم کامپیوتری منابع محاسباتی خود را، برای حل سریعتر مسائل، ترکیب کنند.

openMosix یک کلاستر مبتنی بر لینوکس است که با توزیع یکنواخت کارها روی سیستم‌های موجود، کارایی اجرای برنامه ها را افزایش می دهد.

مزیت اصلی openMosix این است که تمام برنامه ها به طور شفاف و بدون نیاز به استفاده از کتابخانه های خاص، قادر به استفاده از محیط محاسباتی توزیع شده openMosix خواهند بود و نیازی به تغییر در ساختار برنامه ها وجود ندارد.

در این گزارش نحوه نصب، استفاده و ساختار درونی openMosix تشریح شده است.

۲- تعریف کلاستر

در تعریف کلی یک خوشه یا کلاستر مجموعه ای از اجزایی است که به طور مستقل از یکدیگر کار کرده و توسط یک واسطه هماهنگ می شوند.

۲-۱- تعریف کلاسترکامپیوتري

کلاستر کامپیوتری گروهی از کامپیوترهای مجزا است که با یکدیگر کار می کنند، به گونه ای که می توان آنها را تقریبا یک کامپیوتر واحد دانست. اجزای یک کلاستر معمولا به وسیله شبکه های محلی پرسرعت به یکدیگر متصلند.

۲-۲- تقسیم بندي کلاسترهاي کامپوتري

کلاسترها به سه دسته کلی تقسیم می شوند:

- HAC (High Availability Cluster): در این کلاستر هدف افزایش آماده بودن [1] سرویس هائیست هائی که کلاستر فراهم می کند، به وسیله ندهای تکراری، می باشد. هنگام خراب شدن یکی از ندها، ندهای دیگر به سرویس دهی ادامه می دهند و کاربر متوجه اشکال به وجود آمده نمی شود.
- LBC (Load Balancing Cluster): در این نوع کلاسترها تمام بار کاری به یک یا چند سرور ارجاع شده و آنها بار را میان تعدادی سرور back-end تقسیم می کنند. این نوع سرورها معمولا خصوصیت HAC را نیز دارند. به

[1] Availability

چنین کلاسترهایی Server Farm نیز گفته می شود. از جمله پیاده سازی های تجاری LBC می توان Sun Grid Engine را نام برد.

- HPC (High Performance Cluster) : این کلاستر به وسیله تقسیم یک وظیفه میان چندین ند کارائی را افزایش می دهد و بیشتر در محاسبات علمی کاربرد دارد. معمولا HPCها را با استفاده از لینوکس و نرم افزارهای آزاد ایجاد می کنند. به این کلاسترها Beowulf نیز می گویند. این کلاسترها برنامه های خاصی را-که برای بهره برداری از توازی سازی HPC طراحی شده اند- اجرا می کنند. این برنامه ها از کتابخانه هایی مانند MPI استفاده می نمایند.

۳- معرفي openMosix

openMosix یک توسعه کرنل لینوکس برای ساخت کلاسترهای HPC است. openMosix امکان می دهد چندین تک پردازنده و یا تعدادی چندپردازنده[2] که همه از کرنل یکسان استفاده می کنند، با یکدیگر همکاری نمایند. الگوریتم های به اشتراک گذاری openMosix به گونه ای طراحی شده اند که بتواند به تغییرات استفاده از منابع در ندها، در زمان اجرا، پاسخ دهند. این کار به وسیله مهاجرت[3] پروسه ها از یک ند به ند دیگر، به طور شفاف[4] و به منظور توزیع بار و جلوگیری از کوبیدگی[5] ناشی از مبادله صفحات حافظه صورت می گیرد. هدف بهبود کارایی کل کلاستر و ایجاد محیطی چندکاربره و اشتراک زمانی برای برنامه های موازی و متوالی است.

کلاسترهای openMosix می تواند شامل تعداد محدودی از PCها که با سرعت10 MBPS به یکدیگر متصلند باشد و یا مجموعه وسیعی از سرورهای SMP که به وسیله Gigabit LAN و یا ATM به هم مرتبط اند.

بعد از نصب openMosix بر روی تعدادی ند، ندها شروع به صحبت با یکدیگر می نمایند. یک پروسه بر روی هر ند دلخواه می تواند ایجاد شود و openMosix در صورت نیاز آن را به ندی که مناسب تر باشد منتقل می نماید.

---

[2] SMP
[3] Migration
[4] Transparent
[5] Thrashing

مزیت اصلی openMosix این است که چون تغییرات به کرنل اعمال می شود، برنامه ها به طور شفاف و خودکار قادر به استفاده از محیط محاسباتی توزیع شده openMosix خواهند بود و نیازی به تغییر در ساختار برنامه وجود ندارد.

٤- نصب openMosix

openMosix شامل یک patch کرنل و تعدادی ابزار در سطح کاربری است. patch کرنل برای آن است که کرنل بتواند با دیگر ماشین‌های با قابلیت openMosix روی شبکه ارتباط برقرار کند. اگر از بسته RPM برای نصب openMosix استفاده می‌کنیم دیگر نگران patch کرنل نیستیم، زیرا در بسته RPM کرنل patch شده و با گزینه ها و ماژول‌های معمول کامپایل شده است.

ابزارهای سطح کاربر برای استفاده بهینه از قابلیت‌های کرنل کامپایل شده با openMosix لازم هستند، برای مثال برای شروع و خاتمه دادن به سرویس مهاجرت پروسه‌ها (migration)، به کارگیری سیستم فایل openMosix و مهاجرت یک پروسه به ند خاص در شبکه.

بنابراین طبق آن‌چه گفته شد لازم است برای ایجاد یک کلاستر openMosix، دو بسته مختلف را دانلود و نصب کرد: کرنل و ابزارهای سطح کاربر. در این راستا می‌توان دو بسته باینری و یا سورس patch کرنل و ابزارها را دانلود نمود. Patch کرنل بدین صورت نام گذاری می‌شود: openMosix-x.y.z-w که در آن x.y.z نشان دهنده نسخه کرنل موردنظر است که patch باید به آن اضافه شود و w نسخه بازبینی‌شده patch برای همان نسخه کرنل می باشد.

٤-١- کامپایل کردن کرنل لینوکس

ابتدا patch متناسب با سورس کرنلی که در اختیار دارید را دانلود کنید، مثلا برای سورس linux-2.4.16 نیاز به فایل openMosix-2.4.16.x.tar.gz دارید.

اکنون باید patch را به سورس کرنل اضافه و کرنل حاصل را کامپایل نمایید. ابتدا یک کپی از سورس لینوکس خود بگیرید تا در صورت بروز با مشکل در اضافه کردن patch سورس صحیح را از دست ندهید!

```
# cd /usr/src
# mv linux linux.old
```

حال سورس جدید را باز کنید:

```
# tar xzvf /path/to/linux-2.4.16.tar.gz
```

اکنون وارد شاخه سورس شوید و patchای که دانلود کرده‌اید را با دستور زیر به سورس کرنل اضافه کنید:

```
# cd linux
# cat /path/to/openMosix-2.4.16-3.gz | gzip -d | patch -p1 –l
```

پس از این مرحله کرنل آماده کامپایل شدن است. طبق مراحل معمولی که در کامپایل کردن کرنل طی می‌شود می‌توانید این کرنل را کامپایل کنید. تنها تفاوتی که در این‌جا وجود دارد مربوط به تنظیمات openMosix است که به کرنل اضافه شده است. بنابراین در مرحله‌ای که تنظیمات را با اجرای دستور make menuconfig دنبال می‌کنید، حداقل گزینه‌های زیر را فعال کنید:

```
[*] openMOSIX process migration support
[*] Stricter security on openMOSIX ports
[*] Direct File-System Access
[*] openMOSIX File-System
[*] Poll/Select exceptions on pipes
```

بعد از این که کامپایل کرنل با موفقیت انجام شد، باید ابزارها را نصب کنید.

٤-٢- نصب و تنظیم ابزارها

نصب ابزارها به راحتی با کمک فایل راهنمایی که برای نصب در بسته ابزار وجود دارد انجام می‌شود. برای این که نخواهیم ابزارها را روی یک به یک ندهای کلاستر جداگانه نصب کنیم، یک بار آن‌ها را روی یک ند که کرنل همراه openMosix روی آن نصب شده است نصب می‌کنیم، سپس از فایل‌های حاصل از نصب یک فایل tar می‌سازیم. حال کافی است این فایل tar را روی هر ند باز کنیم.

برای ساخت این فایل tar روند زیر را دنبال کنید:

فرض کنید که روی یک ماشین کرنلِ همراه openMosix را در شاخه /usr/src/linux نصب کرده‌اید. فایل سورس ابزار را باز کنید و وارد شاخه مربوطه شوید:

```
# cd /tmp
# tar xzvf /path/to/openMosixUserland-0.1.3.tgz
# cd openMosixUserland-0.1.3
```

حال فایل تنظیمات را به درستی تنظیم کنید. تنظیمات باید مشابه زیر باشد:

```
OPENMOSIX=/usr/src/linux
PROCDIR=/proc/mosix
MONNAME=mmon
```

```
CC=gcc
INSTALLDIR=/usr
CFLAGS=-I/m/include -I./ -I/usr/include -I\$(OPENMOSIX)  /include
-O2
INSTALL=/usr/bin/install
```

ابزار را کامپایل کنید:

```
# make clean build man
```

حال ابزار را در یک شاخه موقتی مانند زیر نصب کنید:

```
# mkdir /tmp/openmosix-tools
# make INSTALLDIR=/tmp/openmosix-tools/usr install
# mkdir /tmp/openmosix-tools/etc
```

اکنون یک تصویر از تمام ابزارهای openMosix **در** /tmp/openMosix-tools **نصب شده است. می‌توان آن را به راحتی در شاخه / از هر ند باز کرد. قبل از آن یک سری تغییرات دیگر هم در فایل** tarball **می‌دهیم: یک اسکریپت برای** startup/shutdown **و یک فایل تنظیمات.**

ابتدا فایل /etc/mosix.map **را بسازید که فایل تنظیمات کلاستر است:**

```
vi /tmp/openmosix-tools/etc/mosix.map
```

حال یک خط برای هر ند در کلاستر مانند زیر اضافه کنید:

```
<node number>  <ip address>  <span>
```

node number **را می‌توانید از شماره ۱ برای ندها شروع کنید. مقدار** IP Adress **را هم برای هر ند تعیین کنید و در نهایت مقدار "1" را برای** sapan **قرار دهید. مثلا برای یک کلاستر با دو ند فایل** map **می‌تواند مانند زیر باشد:**

```
1   192.168.1.1   1
2   192.168.1.9   1
```

اگر بخواهید کلاستری با ۱۰۰ ند داشته باشید که آدرس ندها از 192.168.1.1 **تا** 192.168.1.100 **است، با استفاده از گزینه** span **می توان این کار را انجام داد:**

```
1 192.168.1.1 100
```

سپس فایل map را ذخیره کنید.

اکنون اسکریپتی برای فعال و غیرفعال کردن openMosix می‌نویسیم. ابتدا در شل دستور زیر وارد کنید و سپس فایل openMosix را بسازید:

```
# install -d /tmp/openmosix-tools/etc/rc.d/init.d
#vi /tmp/openmosix-ools/etc/rc.d/init.d/openmosix
```

اسکریپت زیر را در آن کپی کنید:

```
#!/bin/bash
   case $1 in
   stop)
    echo "Stopping openMosix"
    echo 0 > /proc/mosix/admin/mospe
    rm -f /var/lock/subsys/mosix
    ;;
   start)
        echo "Starting
           openMosix"
    if [ -s /etc/overheads -a -f /proc/mosix/admin/overheads
           ]
    then
    grep -v '^#' /etc/overheads >
           /proc/mosix/admin/overheads
    fi
    if [ -s /etc/mfscosts -a -f /proc/mosix/admin/mfscosts ]
    then
    grep -v '^#' /etc/mfscosts >
           /proc/mosix/admin/mfscosts
    fi
    local a1
    local a2
    a1=[ -s /etc/mospe ] && a1="-p `cat /etc/mospe`"
    a2=[ -s /etc/mosgates ] && a2="-g `cat /etc/mosgates`"
    setpe -W $a1 $a2 -f /etc/mosix.map
    touch /var/lock/subsys/mosix
    ;;
   esac
```

در نهایت فایل اسکریپت را با دستور زیر قابل اجرا کنید:

```
#chmod +x /tmp/openmosix-tools/etc/rc.d/init.d/openmosix
```

هر ند باید یک شاخه برای mount کردن سیستم فایل داشته باشد، بنابراین شاخه mnt/ را اضافه کنید:

```
# mkdir /tmp/openmosix-tools/mfs
```

اکنون شاخه نهایی شده است و آماده ساخت tarball هستید:

```
# cd /tmp/openmosix-tools
# tar czvf /tmp/openmosix-tools.tar.gz
```

حال یک فایل openmosix-tools.tar.gz در شاخه tmp/ دارید که می‌توانید آن را روی هر ند نصب کنید:

```
# tar xzpvf /path/to/openmosix-tools.tar.gz -C /
```

قبل از reboot کردن ندها، در هر ند در شاخه etc/fstab/ خط زیر را اضافه کنید تا فایل سیستم openMosix در mfs/، mount شود. همچنین openMosix را به سرویس‌های قابل اجرا در زمان boot اضافه کنید:

```
#cp / openmosix-tools/etc/rc.d/init.d/openmosix /etc/rc.d/init.d/
#Chkconfig /etc/rc.d/init.d/openmosix on
```

با restartکردن تمام ندها، کلاستر شما آماده است!

حال اگر به یکی از ندها login کنید و دستور mmon را وارد کنید یک نمودار از بار حاضر روی کلاستر برای شما روی صفحه نمایش داده می‌شود.

برای اجرای پروسه‌ها و مدیریت مهاجرت آن‌ها و دیگر امکانات openMosix باید از دستورات مناسب آن استفاده کنید. البته با نصب openMosixview از درگیرشدن با دستورات انبوه رهایی می‌یابید! در ادامه این ابزار مفید شرح داده می‌شود.

۵- تکنولوژي openMosix

تکنولوژی openMosix شامل دو بخش است:

- **مکانیزم مهاجرت پروسه قبضه کننده[6]**
- **مجموعه ای از الگوریتم های اشتراک منابع**

[6] Preemptive Process Migration

هر دو بخش در سطح کرنل، با استفاده از یک ماژول قابل لود پیاده سازی شده اند، به گونه ای رابط کرنل[7] تغییر نمی کند و برای برنامه های کاربردی کاملا شفاف است.

۵-۱- مکانیزم مهاجرت پروسه (PPM)

PPM می تواند هر پروسه را در هر زمان و به هر ند آماده منتقل کند. معمولا مهاجرت ها بر اساس اطلاعاتی است که توسط یکی از الگوریتم های به اشتراک گذاری منابع فراهم می شود، اما کاربران می توانند به طور دستی پروسه ها را به ندهای دلخواه منتقل کنند.

هر پروسه یک ند خانگی یکتا به نام UHN [8] دارد که بر روی آن ایجاد می شود. تمام پروسه هایی که روی یک ند ایجاد می شوند، محیط اجرائی UHN را به اشتراک می گذارند. پروسه هایی که به ندهای دیگر منتقل می شوند تا حد امکان از منابع ند محلی (ند دور) استفاده می کنند، اما به وسیله UHN با محیط اصلی تعامل دارند.

تا زمانی که درخواست ها برای منابع (پردازنده، حافظه، ...) کمتر از مقدار آستانه[9] خاصی باشد، پروسه های کاربر محدود به UHN هستند. بعد از گذشتن از آستانه، بعضی پروسه ها به ندهای دیگر منتقل می شوند. هدف کلی حداکثر کردن کارایی با بهره برداری از کل منابع موجود در کلاستر است.

واحد توزیع کار در openMosix پروسه می باشد و در حال حاضر openMosix قادر به توزیع نخ های[10] یک پروسه نیست.

هنگامی که یک پروسه ایجاد می شود، openMosix آن را به آماده ترین ند اختصاص می‌دهد. اگر در طول اجرا منابع جدیدی آماده شدند، الگوریتم های اشتراک منابع ممکن است محل پروسه را تغییر دهند.

در openMosix کنترل مرکزی وجود ندارد، هر ند به عنوان یک سیستم خودمختار عمل می کند و تصمیمات کنترلی را مستقلا اتخاذ می نماید. ندها می توانند به راحتی کلاستر را ترک کنند یا به آن بپیوندند. به علاوه عدم وجود کنترل مرکزی قابلیت توسعه تعداد ندها را فراهم می کند. قابلیت توسعه به وسیله به کارگیری

---

[7] Kernel Interface
[8] Unique Home-Node
[9] Threshold
[10] Thread

تصادف[11] در الگوریتم های کنترلی فراهم می شود. هر ند تصمیماتش را بر اساس اطلاعات جزئی که از سایر ندها دارد می گیرد و نمی کوشد که وضعیت جامع کلاستر را تعیین کند. برای مثال در الگوریتم "توزیع اطلاعات احتمالی"[12] هر ند به طور منظم اطلاعات منابع آزاد خود را به تعدادی از ندها که به طور تصادفی انتخاب می شوند ارسال می کند.

۵-۲- الگوریتم هاي اشتراك منابع

مهمترین الگوریتم های اشتراک منابع در openMosix، موازنه باز[13] و راهنمای حافظه[14] هستند.

۵-۲-۱- الگوریتم موازنه بار

این الگوریتم پیوسته سعی می کند با مهاجرت پروسه ها از ندهای با بار بیش تر به ندهای با بار کمتر، اختلاف بار میان ندها را کاهش دهد. این الگوریتم غیر متمرکز است و تمام ندها الگوریتم یکسانی را اجرا می کنند. تعداد پروسه ها در هر ند و سرعت ندها دو پارامتر مهم برای الگوریتم توزیع بار هستند. الگوریتم به تغییرات بار ندها و خصوصیات زمان اجرای پروسه ها پاسخ می دهد.

مدل ریاضی این الگوریتم بر اساس تحقیقات انجام شده در حوزه اقتصاد است. تعیین محل بهینه برای یک کار مسئله پیچیده ایست، زیرا منابع آماده در کلاستر ناهمگون[15] هستند. هزینه های حافظه، پردازنده، ارتباطات پروسه ها، ... غیرقابل مقایسه اند و حتی بر حسب واحدهای یکسان اندازه گیری نمی شوند. منابع ارتباطی برحسب پهنای باند، حافظه برحسب فضا و پردازنده برحسب سیکل اندازه گیری می شوند. الگوریتمی که توسط openMosix به کار گرفته می شود سعی می کند این تفاوت ها را - بر اساس قوانین اقتصادی و تحلیل رقابتی[16]- با یکدیگر تطبیق دهد.

ایده اصلی تبدیل هزینه های متفاوت استفاده از منابع ناهمگون به یک هزینه همگون[17] واحد است. بر اساس نتیجه الگوریتم، پروسه به ماشین با کمترین هزینه نسبت

---

[11] Randomness
[12] Probabilistic Information Dissemination
[13] Load-balancing
[14] Memory Ushering
[15] Heterogeneous
[16] Competitive Analysis
[17] Homogeneous

داده می شود و الگوریتمی تقریبا بهینه برای تخصیص و به اشتراک گذاری منابع است.

openMosix در زمان boot up، سرعت هر ند را برحسب سرعت یک پردازنده Pentium 1GHz محاسبه و در متغیری به نام speed قرار می دهد. الگوریتم توزیع بار از speed در محاسبات خود استفاده می کند. این متغیر را می توان به طور دستی در proc/hpc/ تنظیم نمود.

۵-۲-۲- الگوریتم راهنماي حافظه

این الگوریتم پروسه ها را به گونه ای در حافظه کلاستر[18] جای می دهد که تا حد امکان از کوبیدگی ناشی از تعویض صفحات حافظه (swapping) جلوگیری شود.

الگوریتم هنگامی راه اندازی[19] می شود که یک ند شروع به تعویض مدام صفحات حافظه به خاطر کمبود فضا می کند. در این حالت الگوریتم راهنمای حافظه، الگوریتم توزیع بار را لغو[20] و سعی می کند پروسه را به ندی که حافظه کافی دارد منتقل نماید، حتی اگر منجر به توزیع بار غیریکنواخت شود.

۵-۳- مهاجرت پروسه(PPM)

openMosix از مهاجرت پروسه به طور قبضه شونده و کاملا شفاف پشتیبانی می کند. بعد از مهاجرت پروسه به تعامل با محیطش بدون در نظر گرفتن مکان جدید ادامه می دهد.

برای پیاده سازی PPM پروسه به دو بخش تقسیم می شود: زمینه کاربر[21] که می تواند بر روی ندها جابه جا شود و زمینه سیستم[22] که وابسته به UHN است و منتقل نمی شود. زمینه کاربر یا remote شامل کد برنامه، داده، نقشه های حافظه و رجیسترهای پروسه است. زمینه سیستم یا deputy شامل شرح منابع وابسته به UHN پروسه و یک استک کرنل برای اجرای کدهای سیستمی از طرف پروسه است. رابط میان remote و deputy کاملا تعریف شده است.

زمان مهاجرت یک بخش ثابت دارد برای ایجاد ساختارهای لازم اجرای پروسه در remote و یک زمان خطی که متناسب با تعداد صفحات حافظه است که

---

[18] Cluster-wide Memory
[19] Trigger
[20] Override
[21] User Context
[22] System Context

منتقل می شود. به منظور حداقل کردن سربار مهاجرت، فقط جداول حافظه و صفحات تغییر یافته[23] پروسه منتقل می شوند.

در زمان اجرای پروسه، فراخوانی های سیستمی پروسه بررسی می شوند. اگر فراخوانی وابسته به UHN باشند به deputy ارسال شده، deputy آن را از طرف پروسه اجرا و نتیجه را برمی‌گرداند. در غیر این صورت بر روی remote اجرا می گردند. این نوع ارتباط میان remote و deputy همگام[24] است.

نوع دیگر تعامل میان remote و deputy ارسال سیگنال - به عنوان مثال هنگام دریافت داده های شبکه در UHN- است. این ارتباط غیرهمگام[25] می باشد.

۵-٤- نکاتي در رابطه با مکانیزم remote/deputy

همان طور که گفته شد deputy نماینده پروسه منتقل شده در UHN است. چون تمام فضای حافظه در ند remote قرار دارد، deputy نقشه حافظه پروسه مهاجرت یافته را نگه نمی‌دارد. در اجرای بسیاری از فراخوانی های سیستمی لازم است که داده بین فضای کاربر و کرنل منتقل شود. در openMosix هنگامی که چنین فراخوانی های سیستمی از remote به deputy ارسال می‌شود، deputy با remote برای انتقال داده ارتباط برقرار می کند. سربار ناشی از کپی کردن داده از remote به deputy می تواند بسیار زیاد باشد. برای بهبود کارایی، یک کش ویژه پیاده سازی شده است که تا حد امکان داده ها را در شروع فراخوانی پیش واکشی[26] می نماید. برای جلوگیری از حذف یا رونویسی صفحات حافظه (هنگام تعویض صفحات) به علت عدم وجود نقشه حافظه برای پروسه منتقل شده، deputy جدول ویژه ای برای اینگونه صفحات حافظه نگه دارد.

پروسه های مهمان و حافظه آنها در ند remote قابل دستیابی برای پروسه های دیگر آن ند و بالعکس نیستند.

ممکن است نیاز باشد یک پروسه در حالتی که متوقف است بعضی توابع openMosix را اجرا کند. به عنوان مثال مهاجرت پروسه می تواند در زمان توقف پروسه اتفاق بیافتد. پروسه می‌بایست تابع را اجرا کرده و مجددا متوقف شود. به این منظور openMosix یک وضعیت منطقی از پروسه ها نگه می دارد که نشان می دهد

---

[23] Dirty Pages
[24] Synchronous
[25] Asynchronous
[26] Pre-Fetching

بقیه پروسه ها وضعیت پروسه مذکور را چگونه باید ببینند و از دیدن حالت اجرای موقتی پروسه جلوگیری می کند.

۵-۵- سیستم فایل openMosix

ساختار بیشتر سیستم های فایل کلاسترها مبتنی بر یک فایل سرور مرکزی است. openMosix از فایل سیستمی به نام DFSA (Direct File System access) استفاده می کند. DFSA برای کاهش سربار ناشی از اجرای فراخوانی های I/O oriented یک پروسه مهاجرت یافته طراحی شده است.

DFSA بر روی سیستم فایل کلاستر اجرا می شود. سیستم فایل هایی که DFSAمی‌تواند بر روی آنها اجرا شود MFS، GFS و QFS هستند.

در حالت عادی هر I/O به UHN برای اجرا فرستاده می شود. اما با استفاده از DFSA اگر فایل مورد نظر بر روی همان ندی باشد که پروسه اجرا می شود، I/O محلی انجام می شود.

علاوه بر DFSA الگوریتم دیگری که عملیات I/O پروسه ها را نیز در نظر می گیرد، به الگوریتم توزیع بار اضافه شده است. در این الگوریتم سعی می شود پروسه به ندی منتقل شود که بیشترین عملیات I/O خود را در آن جا انجام می دهد. در نتیجه برخلاف سیستم فایل های شبکه ای موجود، مانند NSF، که داده را از فایل سرور به ند کلاینت می آورد، کلاستر openMosix سعی می کند پروسه را به ندی منتقل کند که فایل واقعا در آن جا قرار دارد.

۵-٦- APIي openMosix

APIی openMosix در ابتدا به وسیله تعدادی فراخوانی سیستمی پیاده سازی شده بود که برای تنظیم، سوال پرسیدن[۲۷] و راه اندازی openMosix به کار می رفت. به جهت هماهنگی با قرارداد لینوکس، API به گونه ای تغییر یافت که از طریق سیستم فایل مجازی proc/ قابل دستیابی باشد. (proc/ یک سیستم فایل مجازی است که از طریق آن می توان اطلاعات کرنل و پروسه های در حال اجرا را مشاهده کرد.)

در openMosix مسیر proc/ با افزودن proc/openMosix/ توسعه یافته است.از طریق proc/openMosix/ فراخوانی های زیر را می توان انجام داد:

- درخواست مهاجرت پروسه

[27] Query

- قفل کردن یک پروسه برای جلوگیری از مهاجرت
- یافتن محل اجرای فعلی پروسه
- مدیریت سیستم
- کنترل جمع آوری آمار پروسه های در حال اجرا
- اطلاعات در مورد پروسه های remote
- اطلاعات در مورد محدودیت های مهاجرت

٦- معرفي openMosixview

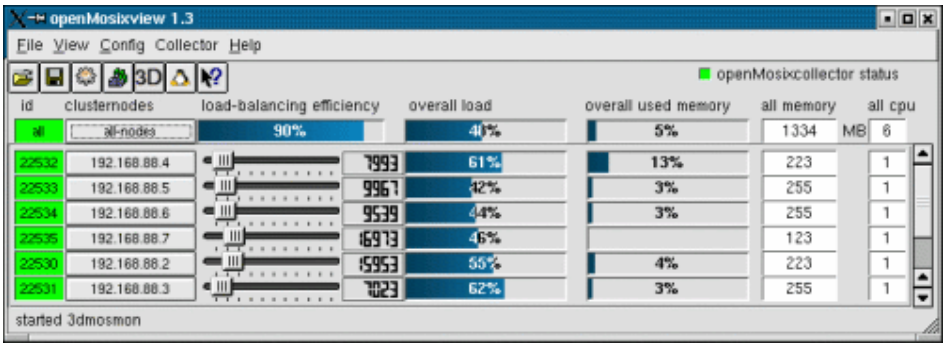
Mosixview یک رابط گرافیکی برای مدیریت کلاسترهای openMosix است. openMosixview نسخه تکمیل شده mosixview می باشد. این بسته شامل ۸ کاربرد مختلف برای نظارت و مدیریت پروسه ها و کلاستر است:

- openMosixprocs : برای مدیریت پروسه ها.
- openMosixcollector : سرویس جمع آوری کننده که اطلاعات ندهای کلاستر را ثبت می کند.
- openMosixanalyzer : برای تحلیل داده های جمع آوری شده توسط ماژول قبل به کار می رود.
- openMosixhistory : تاریخچه پروسه های اجرا شده روی کلاستر.
- openMosixmigmon : جهت مدیریت مهاجرت پروسه ها به کار می رود.
- openMosixpidlog : برای مدیریت یک پروسه خاص به کار می رود.
- 3dmosmon : یک چشم انداز سه بعدی از وضعیت کلاستر.

تمام این کاربردها از صفحه اصلی openMosixview قابل دسترسی است. توسط این بسته مفید تمام اعمالی که از طریق دستورهای مختلف خط فرمان صورت می گیرند، توسط چند کلیک قابل اجرا هستند. در ادامه تعدادی از این ۸ کاربرد توضیح داده می‌شود.

۶-۱- openMosixview

شکل ۱ تصویری از صفحه اصلی openMosixview را مشاهده می کنید.



شکل ۱- صفحه اصلی openMosixview

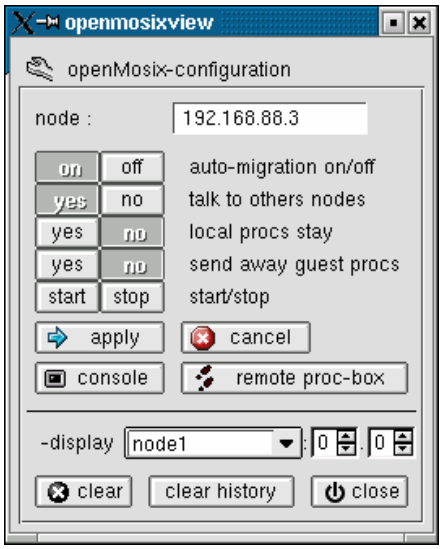
برای هر ند در کلاستر یک سطر حاوی اطلاعات مفیدی راجع به آن ند وجود دارد. در هر سطر اولین خانه چراغی است که openMosix-ld و حالت ند را نشان می دهد، در صورتی که ند در دسترس باشد سبز و در غیر این صورت قرمز است. خانه بعد آدرس IP ند مورد نظر است، اگر بر روی آن کلیک کنید یک دیالوگ تنظیمات ظاهر می شود. sliderی که بعد از آن مشاهده می کنید جهت تنظیم سرعت openMosix برای هر میزبان به کار می رود و خانه بعدی سرعت را به صورت عددی نشان می دهد. با تغییر این مقدار روی چگونگی موازنه بار کل کلاستر تأثیر می‌گذارید. پروسه ها روی کلاستر openMosix بیشتر به ندی که سرعتش بیشتر است مهاجرت می کنند. نکته مهم این است که این مقدار می تواند نشان دهنده سرعت فیزیکی نباشد، بلکه سرعتی است که openMosix برای هر ند در نظر می‌گیرد. روی هر سطر همچنین دو نوار پیشرفتی ملاحظه می‌شود که اولی درصد بار روی ند و دیگری درصد حافظه جاری مورد استفاده ند را نشان می‌دهد. خانه بعدی نشان دهنده کل حافظه در دسترس روی هر ند را است. در انتها تعداد پردازنده در هر ند نوشته شده است.

اولین سطر در صفحه اصلی یک دکمه برای تنظیم تمام ندها دارد. با این امکان، می توان ندها را مدیریت و تنظیم کرد. نوار پیشرفت موجود در سطر اول نشان می‌دهد که کلاستر با چه کیفیتی بار را روی ندها موازنه کرده است. ٪۱۰۰ حالتی است که تمام ندها تقریباً بار یکسان دارند.

در پنجره تنظیمات هر ند (شکل ۲) که از صفحه اصلی قابل دسترسی است، تنظیماتی روی ندها صورت می‌گیرد که از طریق rsh و ssh این تغییرات اعمال می‌شوند. دستوراتی که از این صفحه قابل اجرا هستند شامل موارد زیر می‌باشند:

- automigration on/off
- quiet yes/no
- bring/lstay yes/no

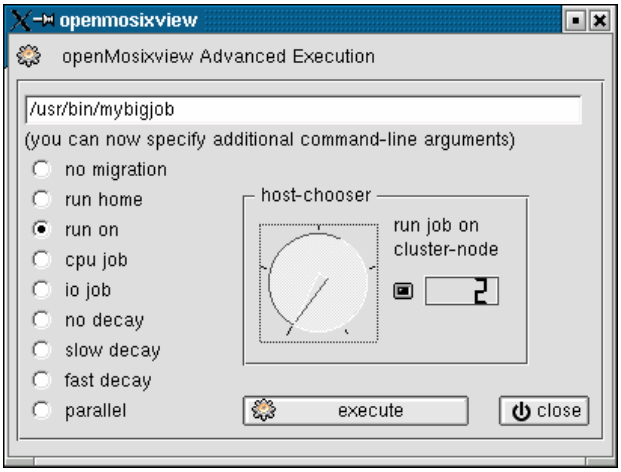
- exspel yes/no
- openMosix start/stop



شکل ۲- پنجره تنظیمات مربوط به هر نود

اجرای پروسه‌ها

دیالوگ Advanced Execution **برای اجرای برنامه‌ها به کار می‌رود. آن را در شکل ۳ مشاهده می‌کنید.**



شکل ۳- دیالوگ **Advanced Execution**

برای بازشدن این پنجره بر آیکون open **از صفحه اصلی کلیک کنید و پروسه موردنظرتان را از شاخه مربوطه‌اش انتخاب کنید. حال می توانید مشخص کنید که**

این پروسه، چگونه و کجا آغاز شود. گزینه های مختلفی برای اجرا وجود دارد. گزینه هایی که در این پنجره وجود دارند عبارتند از:

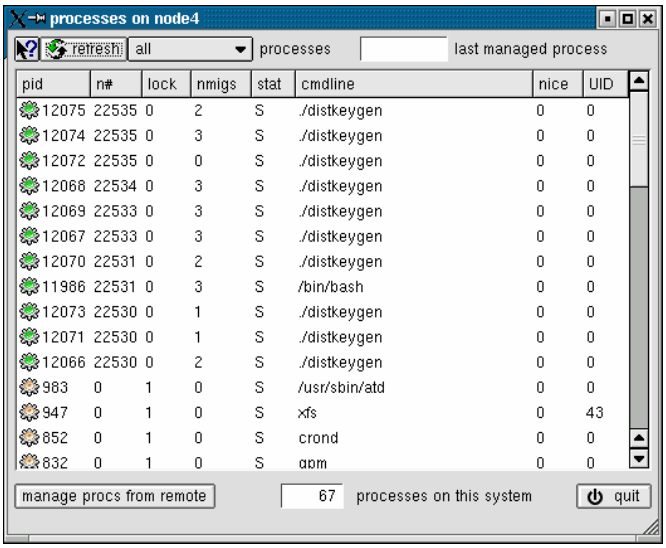
- no migration: **یک کار محلی را آغاز می کند که مهاجرت نخواهد کرد.**
- run home: **یک کار محلی را شروع می کند.**
- run on: **کار را روی ندی که شما از طریق** host-chooser **انتخاب می‌کنید آغاز می‌کند.**
- cpu job: **یک کار با محاسبات بسیار** (computation intensive) **را روی یک ند آغاز می‌کند.** (host-chooser)
- io job: **یک کار با اعمال ورودی/خروجی بسیار** (computation intensive) **را روی یک ند آغاز می‌کند.** (host-chooser)
- parallel: **یک کار را به طور موازی روی چند ند یا تمام ندها آغاز می‌کند.**

البته غیر از این گزینه‌ها می‌توانید آرگومان‌های خط فرمان دیگری را هم در textbox **بالای پنجره مشخص کنید.**

برای اجرای یک پروسه روی ند خاصی، توسط host-chooser **ند مربوطه را مشخص می‌کنیم. این ابزار** ld**ای ندها را نشان می‌دهد و شما می توانید ند موردنظر را بدین‌ترتیب انتخاب کنید. برای اجرای یک پروسه روی چند ند به طور موازی، توسط دو علامتگذاری روی ند اول و آخر، پروسه روی تمام ندهای بین ندهای علامتگذاری شده و خود ندهای علامتگذاری شده اجرا خواهد شد.**

۶-۲- openMosixprocs

دیالوگی که لیست تمام پروسه های روی نود را نشان می دهد از صفحه اصلی openMosixview **قابل دسترسی است. این دیالوگ برای مدیریت پروسه‌هایی که از روی نود حاضر اجرا شده اند و در حال اجرا هستند به کار می رود.**

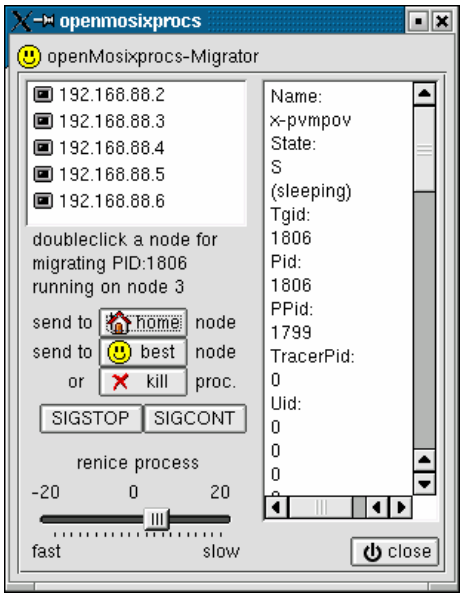


شکل ٤- لیست تمام پروسه های در حال اجرا که از نود حاضر اجرا شده اند.

لیست پروسه‌ها در این دیالوگ نشان می‌دهد که هر پروسه روی چه ندی در حال اجراست. ستون دوم idی ندی را نشان می‌دهد که پروسه روی آن در حال اجراست. شماره 0 بدین معنی است که پروسه روی ند محلی خود در حال اجراست، بقیه شماره‌ها مربوط به ندهای دور هستند. پروسه‌هایی که مهاجرت کرده‌اند با آیکون سبز علامتگذاری شده‌اند.

با دو کلیک روی یک پروسه، پنجره مدیریت مهاجرت پروسه باز می‌شود. (شکل ۵)

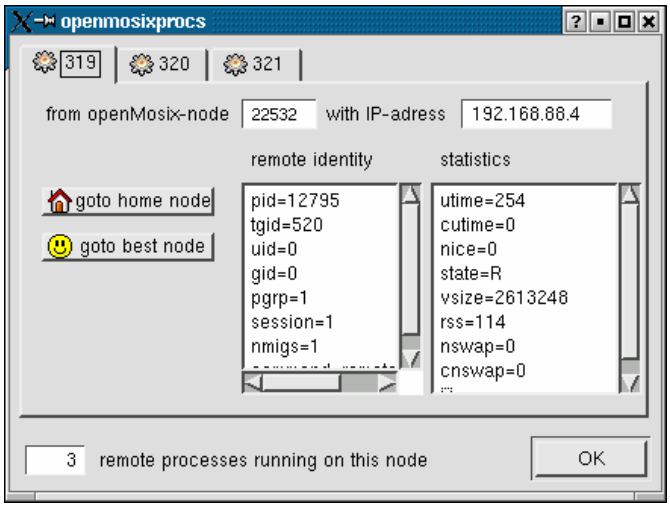
این پنجره تمام ندهای روی کلاستر را نشان می‌دهد و برای مدیریت یک پروسه است. با دوبار کلیک روی یک ند، پروسه به آن ند مهاجرت می‌کند. دکمه home پروسه را به ند محلی خود منتقل می‌کند. دکمه best پروسه را به بهترین ند در دسترس منتقل می‌کند.



شکل ٥ – دیالوگ openMosixprocs

با دکمه kill پروسه از بین می‌رود. برای متوقف‌کردن موقتی پروسه دکمه SIGSTOP و برای اجرای دوباره آن SIGCONT را کلیک کنید.

روی دیالوگ openMosixprocs دکمه دیگری با نام manage procs from remote وجود دارد.



شکل ٦– مدیریت پروسه ها از دور

در پنجره مربوطه (شکل ۶) هر یک از برگه‌های بالای پنجره یکی از پروسه‌هایی است که به ند محلی حاضر مهاجرت کرده است.

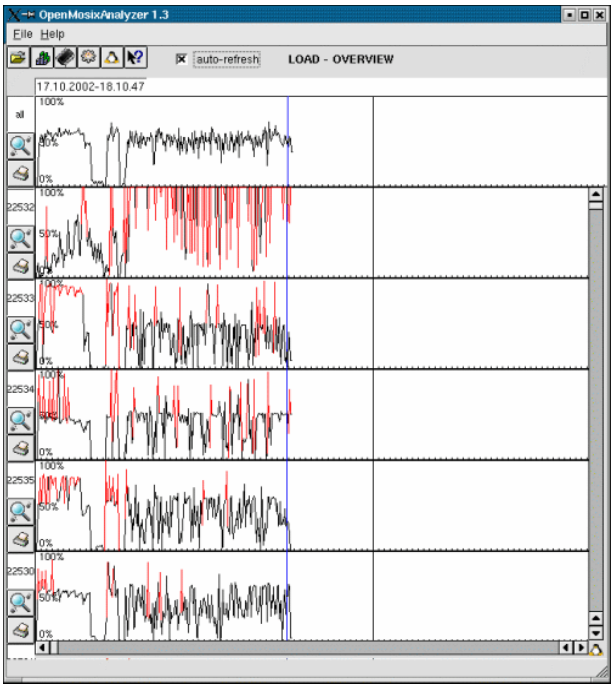
۶-۳- openMosixcollector

این سرویس می تواند روی هر یک از اعضای کلاستر اجرا شود. penMosixcollector اطلاعات مربوط به بار روی هر ند را در شاخه /tmp/openmosixcollector/* ثبت می‌کند. فایل‌های ثبت مربوطه توسط openMosixanalyzer تحلیل و بررسی می‌شوند. openMosixcollector هر ۱۲ ساعت یک بار اجرا می‌شود و تاریخچه حاضر از اطلاعات بار را در /tmp/openmosixcollector[date]/* ذخیره می‌کند. می‌توان یک نقطه بازرسی (checkpoint) در تاریخچه قرار داد. این نقطه‌ها به صورت خط‌های عمودی آبی در رابط گرافیکی openMosixanalyzer علامتگذاری خواهند شد، برای مثال شما می‌توانید وقتی یک پروسه را اجرا می‌کنید یک نقطه بررسی قرار دهید و هنگام خاتمه یک نقطه دیگر. دستورات موجود برای این سرویس عبارتند از:

- openmosixcollector –d : collector را به عنوان یک سرویس اجرا می‌کند.
- openmosixcollector –k : سرویس collector را خاتمه می‌دهد.
- openmosixcollector –n : یک نقطه بازرسی در تاریخچه قرار می‌دهد.
- openmosixcollector -r: تاریخچه حاضر را ذخیره می‌کند و یک تاریخچه جدید را شروع می‌کند.

٦-٤- OpenMosixanalyzer

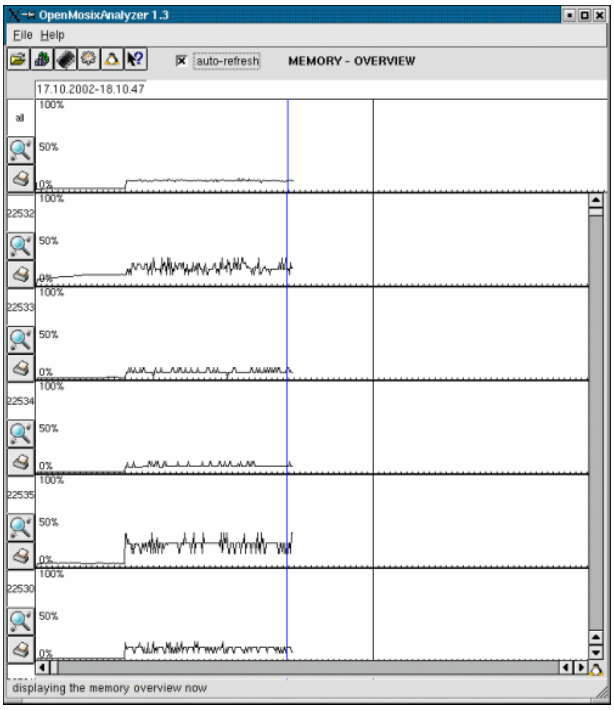
شکل ۷ میزان بار ندها را به طور گرافیکی در openMosixanalyzer نشان می‌دهد.



شکل ۷- دیالوگ openMosixanalyzer ، برگه نمودار میزان بار روی نودها

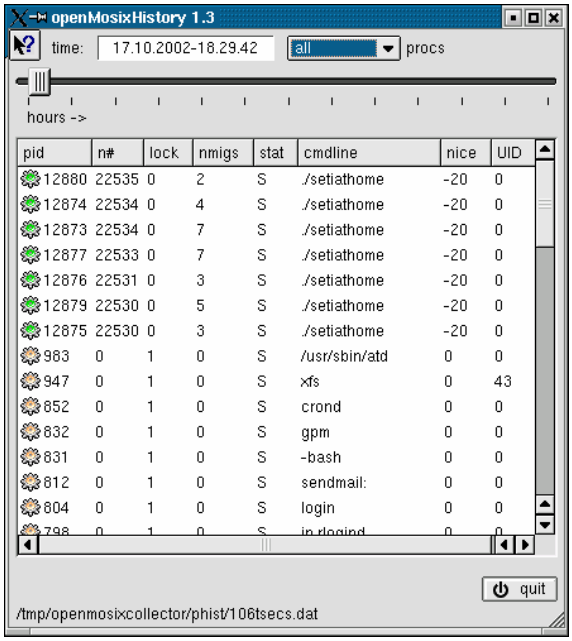
شکلی که نشان داده می‌شود به طور پیش‌فرض از فایل ثبت مربوط به بار روی ندها در زمان حاضر است. در صورتی که بخواهیم تاریخچه‌ای از گذشته را مشاهده کنیم باید فایل ثبت مربوطه آن را از /tmp/openmosixcollector[date]/ انتخاب کنیم. خط های رسم‌شده در نمودارها به طور عادی مشکی هستند، ولی در صورتی که بار بیشتر از ٪۷۵ باشد خط‌ها قرمز رسم شده‌اند. دکمه find out مربوط به هر ند مقادیر آماری مفیدی را برای آن ند محاسبه می‌کند و نمایش می‌دهد.

همچنین از طریق openMosixanalyzer می‌توان نمودارهای مربوط به حافظه مورداستفاده پروسه‌ها درکلاستر را مشاهده کرد.



شکل۸- دیالوگ openMosixanalyzer ، برگه نمودار میزان حافظه مورداستفاده روی نودها

امکان دیگری که در openMosixanalyzer فراهم آمده مربوط به پروسه‌هاست. در این قسمت اطلاعات بسیاری با جزئیات در رابطه با این که چه پروسه ای، در چه زمان و در چه ندی در حال اجرا بوده است به‌دست می‌آید. (شکل ۹)



شکل ۹- دیالوگ openMosixhistory

زمان را می‌توان با نوار پیشرفتی که در بالای دیالوگ وجود دارد تغییر داد.

٦-٥- openMosixmigmon

این برنامه یکی از کاربردهای جالب دیگر از openMosixview است. توسط این قسمت می توانید تمام ندهای حاضر در کلاستر را با پروسه‌های در حال اجرا روی آن‌ها در یک نمای گرافیکی مشاهده کنید. هر ند به صورت یک پنگوئن که در یک دایره نشسته است دیده می‌شود. پنگوئن اصلی ندی است که openmosixmigmon روی آن اجرا شده است. اگر یک پروسه به ندی مهاجرت کند، پروسه از دایره اصلی خود به دایره مربوط به ند دور منتقل می‌شود، سپس پروسه توسط رنگ سبز علامتگذاری می‌شود و یک خط از ند محلی به ند دور برای نشان‌دادن مهاجرت رسم می‌شود. اگر موس را روی پروسه نگه dای پروسه نشان داده می‌شود. همچنین می‌توان به راحتی با کشیدن (drag) یک پروسه توسط موس و رها کردن (drop) آن در ندی دیگر به راحتی پروسه را به ند مربوطه مهاجرت داد. اگر پروسه ای را دوبار کلیک کنید به ند محلی خودش برمی‌گردد.

٧- جمع بندي

با توجه به روند رو رشد کاربردهای محاسباتی سنگین، نیاز به تکنولوژی های پردازش موازی افزایش می‌یابد. از جمله راه حل های موجود کلاسترینگ است.

با استفاده از سیستم عامل لینوکس و openMosix، می توان تعداد دلخواهی از کامپیوترها را تبدیل به کلاستر نمود.

امروزه بیش از ۳۰۰۰ کلاستر openMosix در دنیا وجود دارد. متوسط ندهای کلاسترها ۲۵ عدد است و به راحتی تا ۲۰۰۰ عدد گسترش می یابد.

openMosix دارای جامعه ای فعال و حمایتگر از توسعه دهندگان و استفاده کنندگان از آن می باشد.

منابع

- **OpenMosix, Presented by Dr. Moshe Bar et al. Linux Congress 2003.**
- **OpenMosix, OpenSSI and Kerrighed: A Comparative Study. Renaud Lottiaux Pascal Gallard, Geoffroy Valle, Christine Morin. 2005 IEEE International Symposium on Cluster Computing and the Grid.**
- **Migratable sockets in cluster computing, Kamran Malik, Osama Khan, Tahir Mobashir, Mansoor Sarwar. 2004 Elcevier.**
- **openMosix vs. Beowulf: a case study, Moshe Bar, Stefano Gozzini, Alberto Marmodora.**
- **Wikipedia Encyclopedia: en.wikipedia.org**
- **openMosix's Home: www.openMosix.org**
- **openMosixview Home: www.openmosixview.com**