Controlled Contention Scheduling in Linux Operating System

Presented by
Phanikar Subodh Krishna Chereddi

Overview

- Introduction
- CFS scheduler
- Cgroups
- Managing Cgroups
- Performance Measurement Tools
- Results
- Conclusion

Introduction

- Is scheduling in multicore processor chip a solved problem?
- Modern operating systems like Linux allow both contention and reservation policies.
    - Contention is not good for applications sensitive to interference.
    - Reservation is not good if applications don’t use full time interval.
- This is an attempt to implement policies that tradeoff contention scheduling and gang scheduling in Linux operating system using containers.

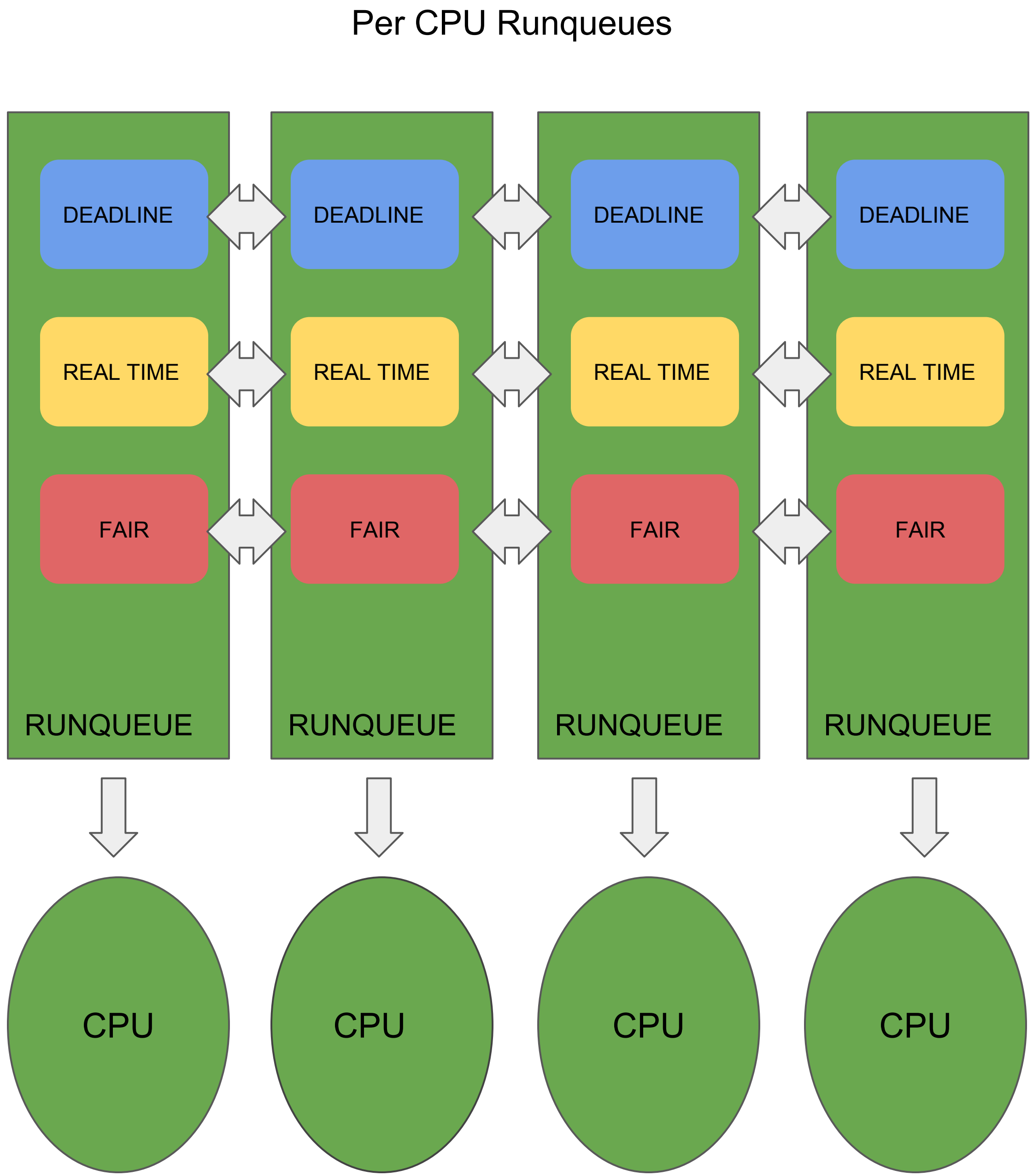
Motivation

“And you have to realize that there are not very many things that have aged as well as the scheduler. Which is just another proof that scheduling is easy.”

- Linus Torvalds

But still :

- Scheduler is undergoing changes in every new release.
- There is no generic scheduler.
- Especially for desktops booting on Linux, measuring the scheduler performance is difficult.



Linux Scheduler Architecture

The are 3 scheduling classes in Linux scheduler, their order of priority is:

1. Deadline
2. Real time
3. Fair

Fair Scheduling in Linux kernel

- The virtual runtime of a task is its actual runtime normalized to the total number of running tasks.

- The process in the run queue with shortest virtual runtime has highest priority to be scheduled.

- CFS runqueue is RB-Tree implementation of a priority queue.

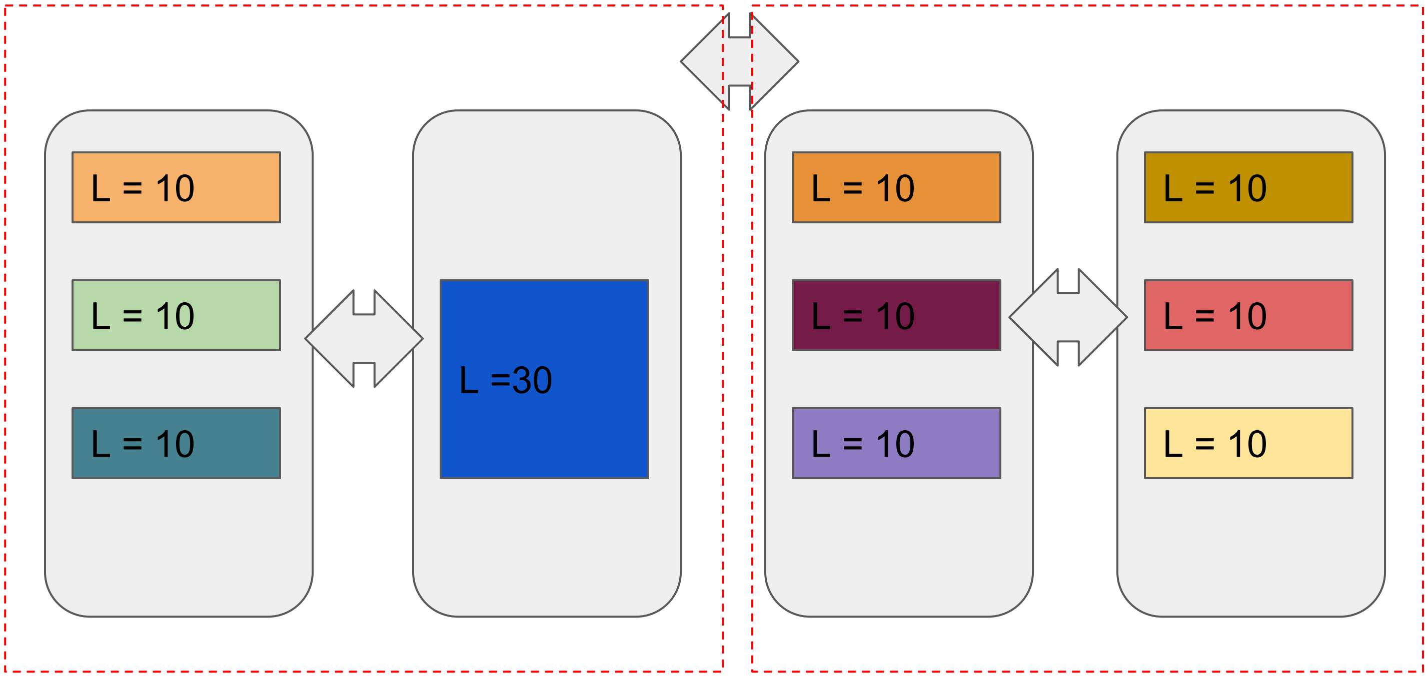
- Each node in RB-Tree is a process and they are self-balanced by virtual runtime.

- The leftmost node is one with least virtual runtime so, It has highest priority.

Load Balancing Among Runqueues

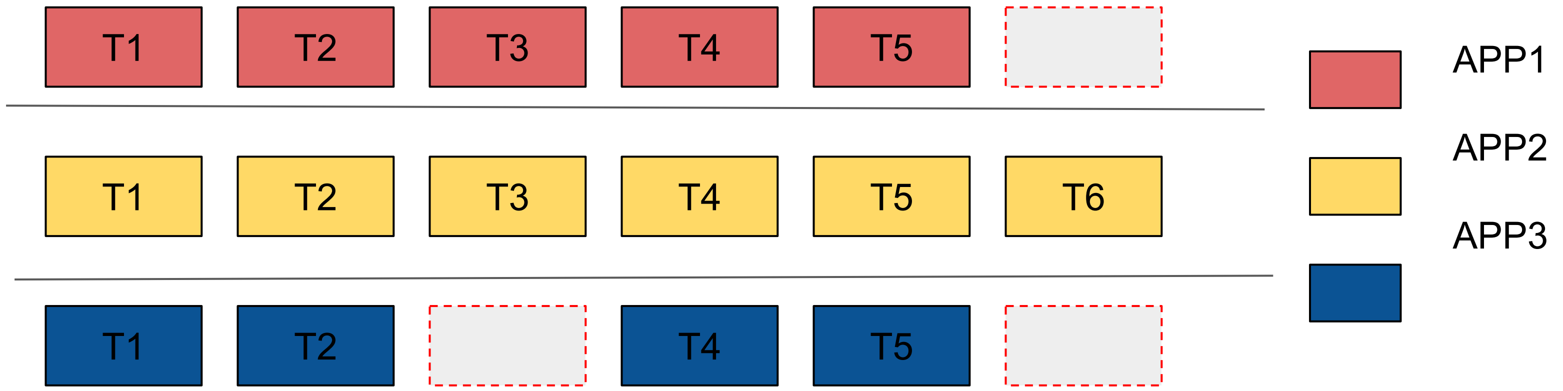
- Load = (Weight of process) * (% Utilization of CPU)

- Load average is metric used to balance the runnable processes among CPUs.



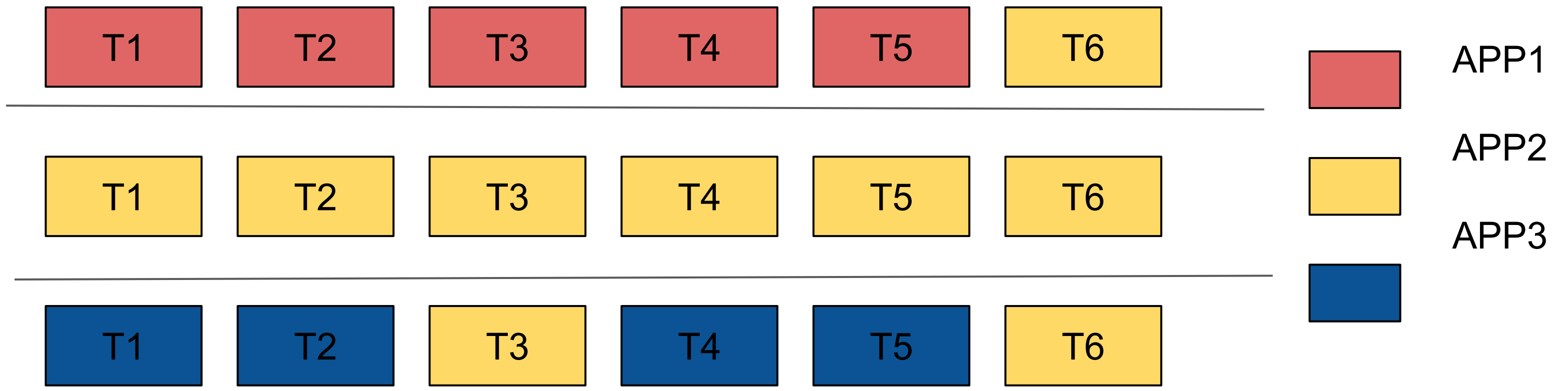
Gang Scheduling

- Strict gang scheduling does not allow threads of other application to be scheduled, may lead to wasted computational power.



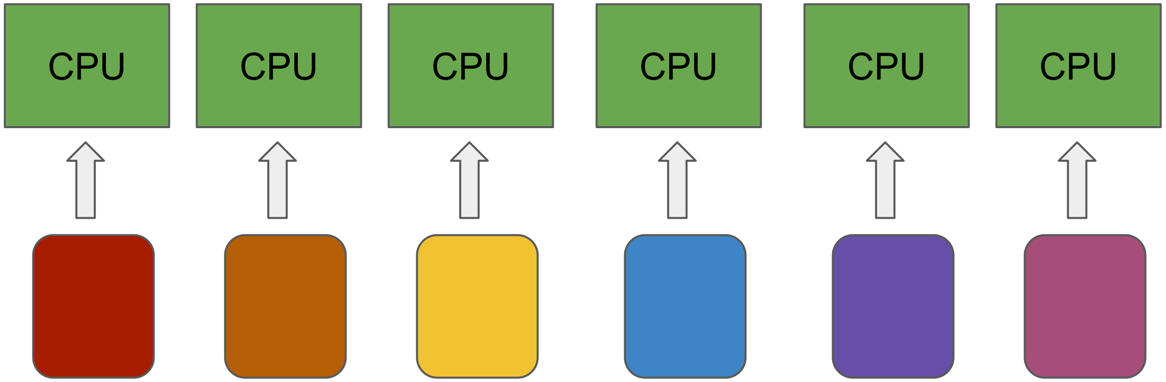
Gang Scheduling

- Allowing other application threads to run when there is an ideal CPU may cause interference.



Pinning Application

- Pining guarantees that application will run only in that CPU.
- It does not mean that only that application will be running exclusively.
- The kernel threads will also be scheduled on all the CPUs.



Containers in Linux Kernel

- Container types:
  - Cgroups
  - Namespaces

- Containers are lightweight virtual machines:
  - Own process space, own network interface, can run stuff as root, can install packages, can run services.

- But a little different from virtual machines:
  - Uses the host kernel, can't boot a different OS, can't have its own modules, doesn't need init as PID 1.

Cgroups

- **blkio** — this subsystem sets limits on input/output access to and from block devices such as physical drives (disk, solid state, or USB).
- **cpu** — this subsystem uses the scheduler to provide cgroup tasks access to the CPU.
- **cpuacct** — this subsystem generates automatic reports on CPU resources used by tasks in a cgroup.
- **cpuset** — this subsystem assigns individual CPUs (on a multicore system) and memory nodes to tasks in a cgroup.
- **devices** — this subsystem allows or denies access to devices by tasks in a cgroup.

Cgroups

- **freezer** — this subsystem suspends or resumes tasks in a cgroup.
- **memory** — this subsystem sets limits on memory use by tasks in a cgroup and generates automatic reports on memory resources used by those tasks.
- **net_cls** — this subsystem tags network packets with a class identifier (classid) that allows the Linux traffic controller (tc) to identify packets originating from a particular cgroup task.
- **net_prio** — this subsystem provides a way to dynamically set the priority of network traffic per network interface.
- **ns** — the *namespace* subsystem.

Cgroups

```
phanikar@phanikar-XPS-8900:~$ ls -l /sys/fs/cgroup/
total 0
dr-xr-xr-x  2 root root  0 Dec  6 10:22 blkio
drwxr-xr-x  2 root root 60 Dec  6 10:22 cgmanager
lrwxrwxrwx  1 root root 11 Dec  6 10:22 cpu -> cpu,cpuacct
lrwxrwxrwx  1 root root 11 Dec  6 10:22 cpuacct -> cpu,cpuacct
dr-xr-xr-x  2 root root  0 Dec  6 10:22 cpu,cpuacct
dr-xr-xr-x 10 root root  0 Dec  6 10:22 cpuset
dr-xr-xr-x  5 root root  0 Dec  6 10:22 devices
dr-xr-xr-x 10 root root  0 Dec  6 10:22 freezer
dr-xr-xr-x  2 root root  0 Dec  6 10:22 hugetlb
dr-xr-xr-x  2 root root  0 Dec  6 10:22 memory
lrwxrwxrwx  1 root root 16 Dec  6 10:22 net_cls -> net_cls,net_prio
dr-xr-xr-x  2 root root  0 Dec  6 10:22 net_cls,net_prio
lrwxrwxrwx  1 root root 16 Dec  6 10:22 net_prio -> net_cls,net_prio
dr-xr-xr-x  2 root root  0 Dec  6 10:22 perf_event
dr-xr-xr-x  5 root root  0 Dec  6 10:22 pids
dr-xr-xr-x  5 root root  0 Dec  6 10:22 systemd
```

Freezer Cgroup

- Allows to **freeze / thaw** a group of processes or threads.
- Similar in functionality to mass SIGSTOP/SIGCONT!!!
- Cannot be detected by the processes (unlike SIGSTOP/SIGCONT).
- Doesn’t impede ptrace/debugging.
- Specific use cases:
  - Cluster batch scheduling
  - Process migration

Freezer Cgroup

- **freezer.state:** Read-write.
  Shows state as THAWED / FROZEN / FREEZING.

- **freezer.self_freezing:** Read only.
  Shows the self-state. 0 if the self-state is “THAWED”; otherwise, 1. This value is 1 iff the last write to freezer.state was “FROZEN”.

- **freezer.parent_freezing:** Read only.
  Shows the parent-state. 0 if none of the cgroup's ancestors is frozen; otherwise, 1.

- **NOTE:** The root cgroup is non-freezable and the above interface files don't exist.

Cpuset Cgroup

- Pin groups to specific CPU(s)
- Reserve CPUs for specific apps
- Avoid processes bouncing between CPUs
- Also relevant for NUMA systems
- Provides extra dials and knobs
    - Per zone memory pressure, process migration costs…

Cpuset Cgroup

- **cpuset.cpus**
  List of the physical numbers of the CPUs on which processes in that cpuset are allowed to execute.

- **cpuset.cpu_exclusive** Flag (0 or 1)
  If set (1), the cpuset has exclusive use of its CPUs (no sibling or cousin cpuset may overlap CPUs).

- **cpuset.sched_load_balance** Flag (0 or 1)
  If set (1, the default) the kernel will automatically load balance processes in that cpuset.

- **cpuset.sched_relax_domain_level -** For NUMA machine
  - Integer, between -1 and a small positive value.
  - The *sched_relax_domain_level* controls the width of the range of CPUs over which the kernel scheduler performs immediate rebalancing.

Cpu,cpuacct Cgroup

- Keeps track of user/system CPU time
- Keeps track of usage per CPU
- Allows to set weights
    - cpu.shares
- How regularly a cgroup's access to CPU resources should be reallocated?
    - cpu.cfs_period_us
- Specifies the total amount of time in which all tasks in a cgroup can run during one period
    - cpu.cfs_quota_us

Managing Cgroups

- Cgroup pesudo filesystem is mounted in **/sys/fs/cgroup/**
- Commands:
  - **cgcreate** -t uid:gid -a uid:gid -g subsystems:path
  - **cgdelete** subsystems:path
  - **cgset** -r parameter=value path_to_cgroup
  - **cgexec** -g subsystems:path_to_cgroup command arguments
  - **lscgroup**

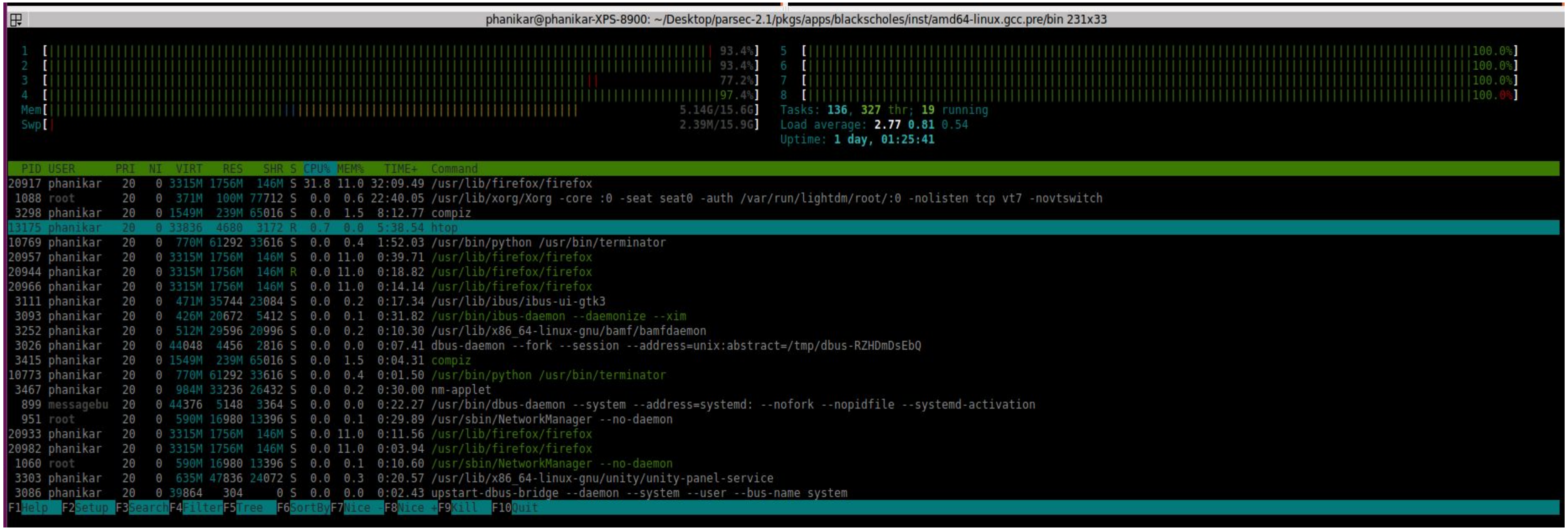
Managing Cgroups

- Manually pushing a PID to a Cgroup
  - echo pid > /sys/fs/cgroup/{subsystem}/{path-to-hierarchy}/tasks

- Writing parameter to configuration files
  - echo THAWED > /sys/fs/cgroup/freezer/app1/freezer.state

Scheduler Analysis / Job Management tools

- Htop
  - Colorful and Interactive
  - Can view resource utilization per process



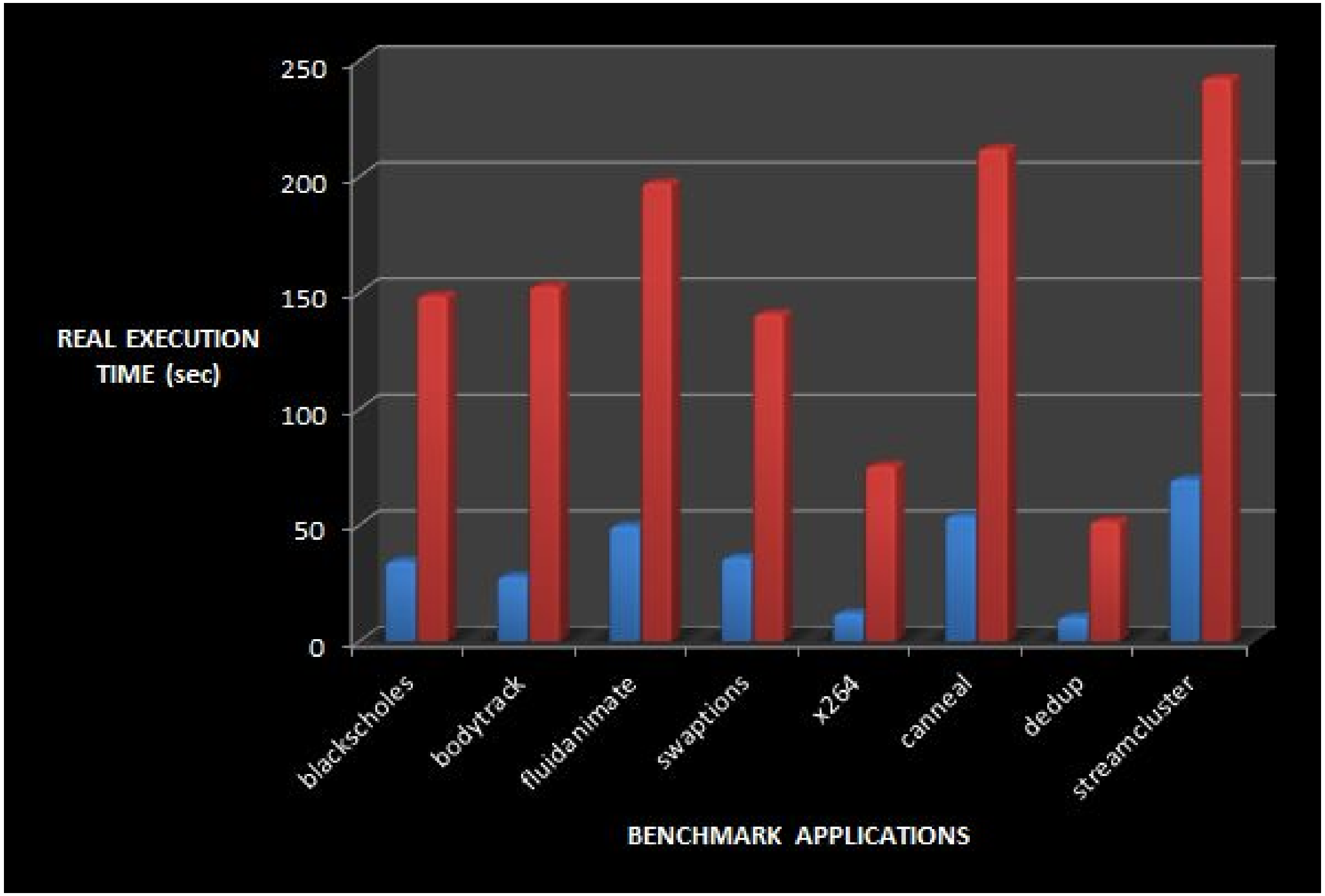
Scheduler Analysis / Job Management tools

- Berkeley Packet Filter (BPF)
  - Runqlat, Cpudist are part of BPF to measure runqueue latency and cpu distribution
  - Easy to debug, written in Python

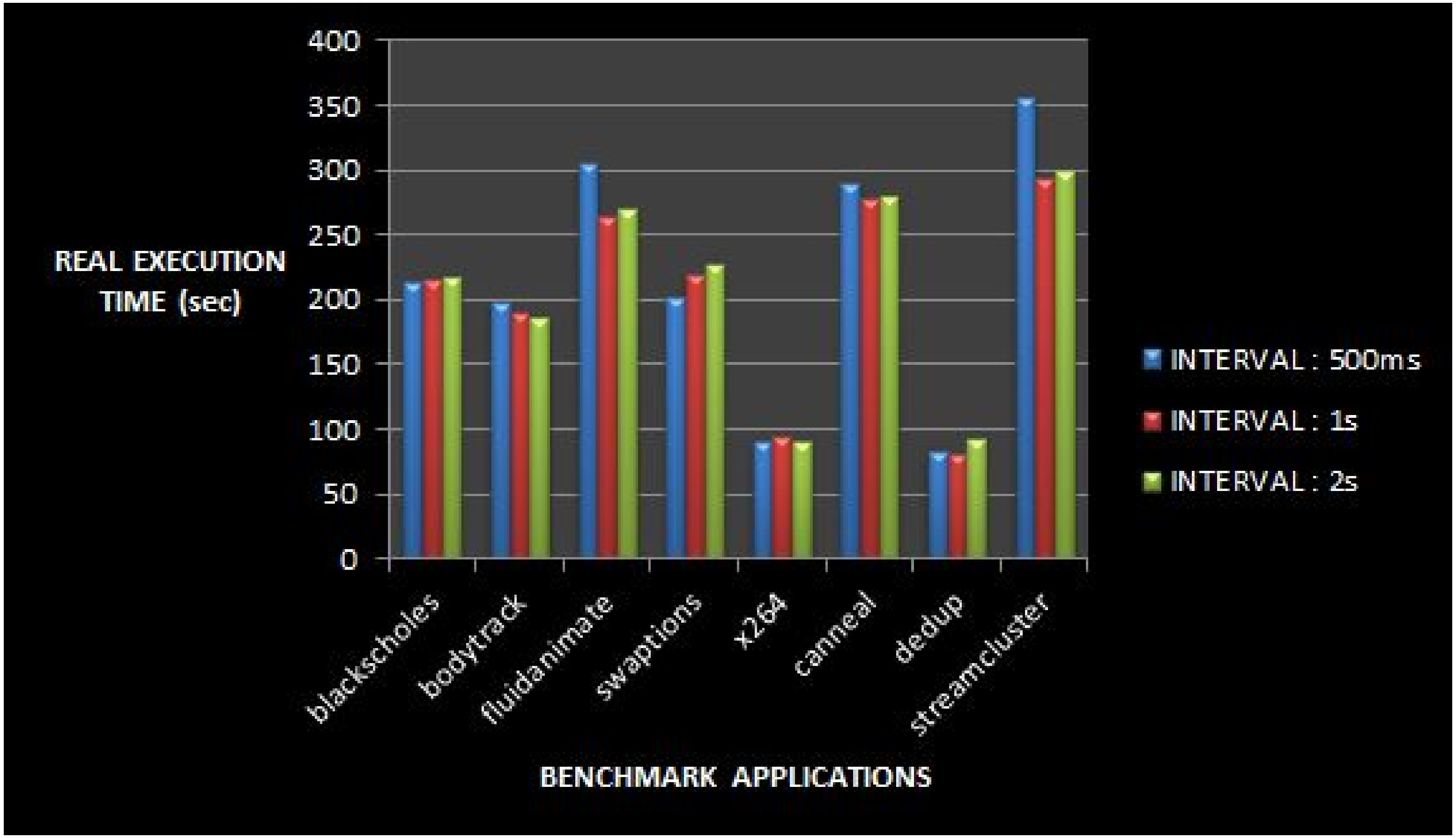
```
root@phanikar-XPS-8900:/usr/share/bcc/tools# ./cpudist
Tracing on-CPU time... Hit Ctrl-C to end.
^C
     usecs               : count     distribution
         0 -> 1          : 161      |***                                     |
         2 -> 3          : 53       |*                                       |
         4 -> 7          : 68       |*                                       |
         8 -> 15         : 66       |*                                       |
        16 -> 31         : 65       |*                                       |
        32 -> 63         : 160      |***                                     |
        64 -> 127        : 147      |***                                     |
       128 -> 255        : 211      |****                                    |
       256 -> 511        : 275      |******                                  |
       512 -> 1023       : 554      |************                            |
      1024 -> 2047       : 388      |*********                               |
      2048 -> 4095       : 1712     |****************************************|
      4096 -> 8191       : 183      |****                                    |
      8192 -> 16383      : 215      |*****                                   |
     16384 -> 32767      : 103      |**                                      |
     32768 -> 65535      : 83       |*                                       |
     65536 -> 131071     : 18       |                                        |
```

Experiment Results

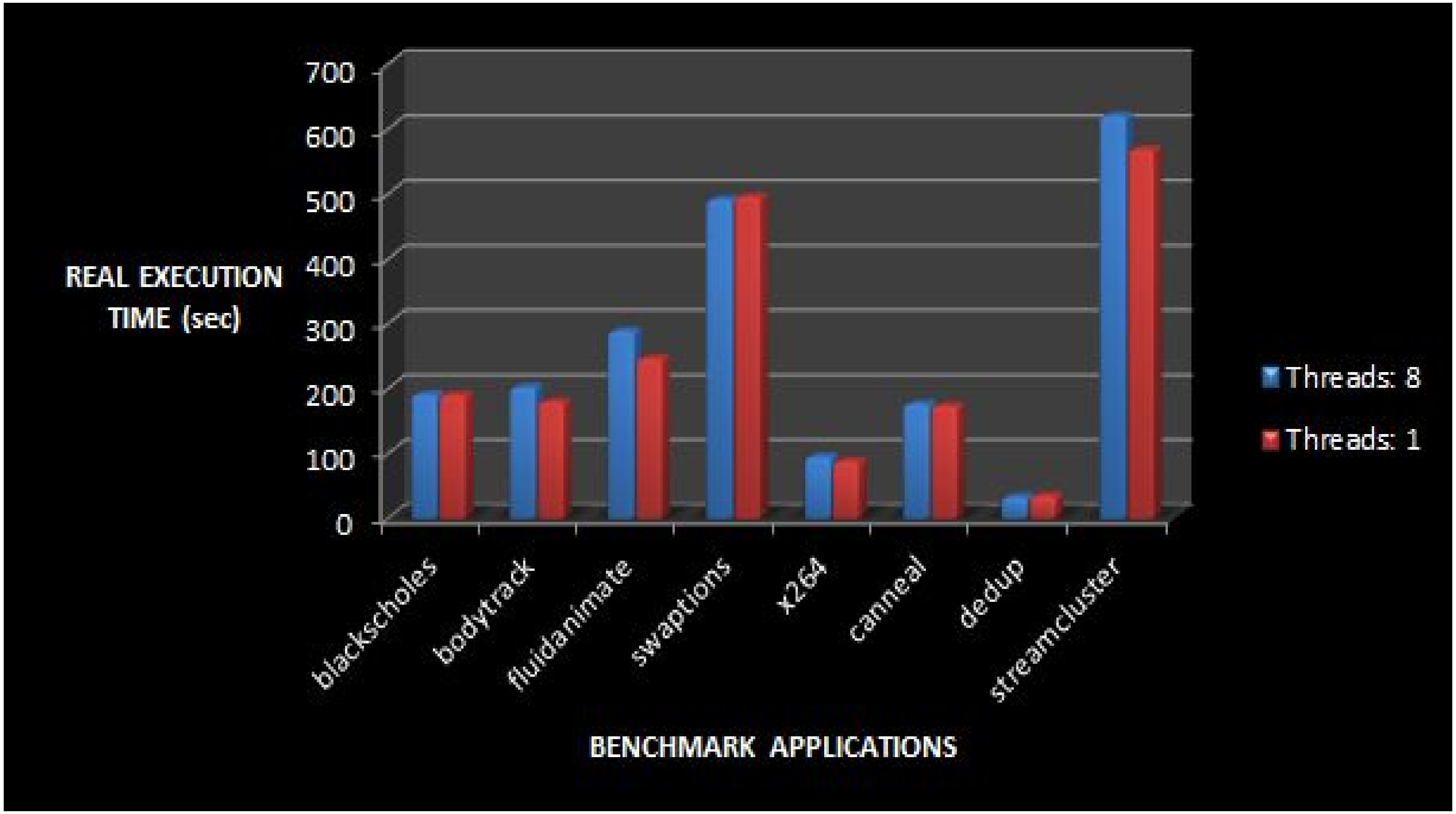
Runtime of Individual Application *VS* Runtime of Applications Running Parallely Using CFS



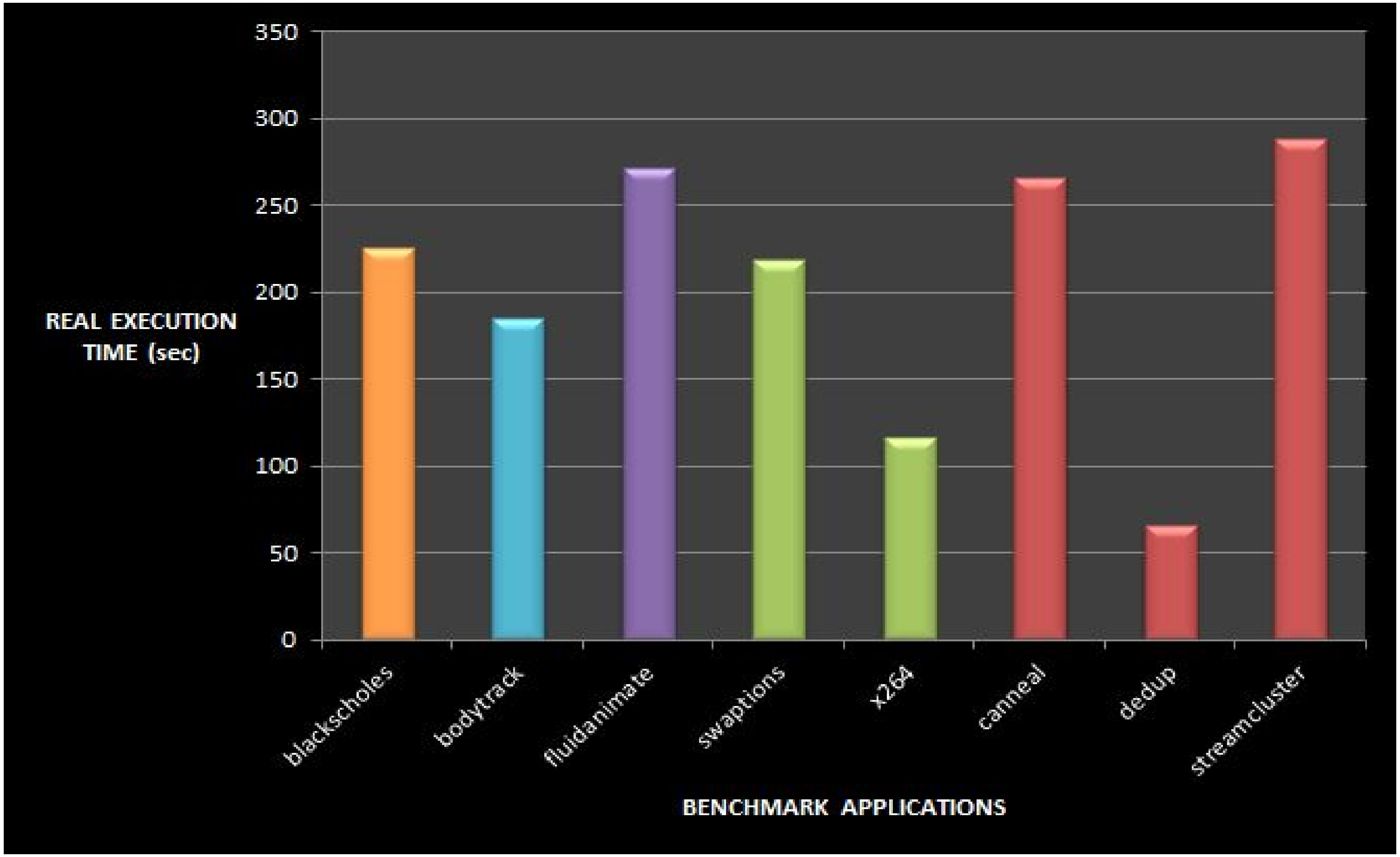
Runtime of Applications Using Gang Scheduling With Different Switching Intervals



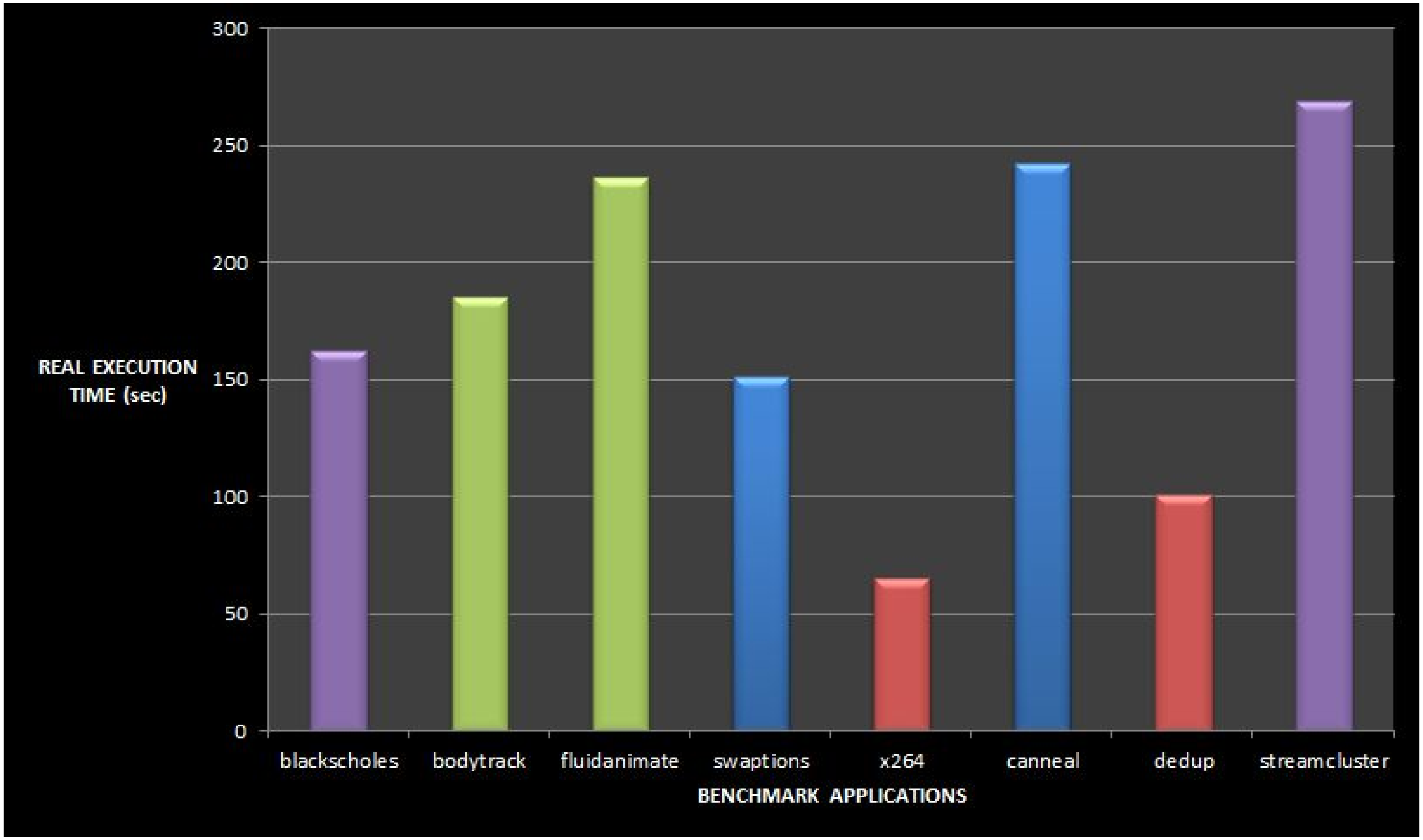
Runtime of Applications Using Pinning With Different Thread Counts



Runtime of Applications When Co Scheduled



Runtime of Applications When Co Scheduled



Conclusions & Future Work

- Terminology used in the kernel leads to confusion often.
- Available books are based on older versions but, there is minor release almost every month.
- There was unexpected behaviour as policy was implemented in userspace instead of kernel space.
- I wish to implement Contention Controlled scheduling policies in the Linux Scheduler in the future.

References

1. Controlled Contention: Balancing Contention and Reservation in Multicore Application Scheduling
2. The Linux Scheduler: a Decade of Wasted Cores
3. extended Berkeley Packet Filters
4. Cgroups

THANK YOU

Running Applications

RUNNING INDIVIDUAL APPLICATION IN SEQUENCE			
application	**real time (sec)**	**virtual time in userspace (sec)**	**threads**
blackscholes	34.345	180.756	8
bodytrack	27.86	178.192	8
fluidanimate	49.598	379.628	8
swaptions	35.933	281.36	8
x264	12.03	87.732	8
canneal	53.803	186.992	8
dedup	10.269	29.724	8
streamcluster	69.606	545.32	8
total	**293.444**	1869.704	

RUNNING ALL APPLICATIONS IN PARALLEL			
application	**real time (sec)**	**virtual time in userspace (sec)**	**threads**
blackscholes	148.689	188.076	8
bodytrack	152.754	176.084	8
fluidanimate	197.296	381.064	8
swaptions	140.818	288.36	8
x264	75.535	87.824	8
canneal	212.055	195.296	8
dedup	51.577	29.084	8
streamcluster	**242.338**	498.644	8
total	**242.345**	1844.432	

RUNNING ALL APPLICATION USING FREEZER			
SWITCHING INTERVAL : 500ms			
application	**real time (sec)**	**virtual time in userspace (sec)**	**threads**
blackscholes	212.874	181.192	8
bodytrack	197.814	179.36	8
fluidanimate	304.145	436.276	8
swaptions	201.32	281.54	8
x264	90.243	88.712	8
canneal	288.907	187.028	8
dedup	82.996	29.512	8
streamcluster	**355.782**	701.66	8
total	**355.782**	2085.28	

RUNNING ALL APPLICATION USING FREEZER			
SWITCHING INTERVAL : 1s			
application	**real time (sec)**	**virtual time in userspace (sec)**	**threads**
blackscholes	216.118	180.432	8
bodytrack	190.264	176.98	8
fluidanimate	264.042	379.636	8
swaptions	218.217	281.4	8
x264	93.331	87.676	8
canneal	276.784	186.756	8
dedup	80.047	29.856	8
streamcluster	**293.788**	545.18	8
total	**293.869**	1868.016	

RUNNING ALL APPLICATION USING FREEZER			
SWITCHING INTERVAL : 2s			
application	**real time (sec)**	**virtual time in userspace (sec)**	**threads**
blackscholes	216.578	181.38	8
bodytrack	185.661	176.728	8
fluidanimate	269.278	379.888	8
swaptions	226.851	281.832	8
x264	90.56	87.888	8
canneal	280.399	186.928	8
dedup	92.934	29.932	8
streamcluster	**298.522**	545.348	8
total	**299.399**	1869.96	

application	real time (sec)	virtual time in userspace (sec)	threads	cpu usage
blackscholes	225.09	188.62	8	83%CPU
bodytrack	185.31	175.74	8	95%CPU
fluidanimate	270.97	391.72	8	144%CPU
swaptions	218.69	290.05	8	132%CPU
x264	115.8	88.3	8	76%CPU
canneal	265.99	218.17	8	82%CPU
dedup	65.69	28.96	8	49%CPU
streamcluster	**288.34**	564.4	8	196%CPU
total	**288.34**	1945.96		

application	real time (sec)	virtual time in userspace (sec)	threads	cpu usage
blackscholes	162.56	189.66	8	116%CPU
bodytrack	185.33	176.22	8	95%CPU
fluidanimate	236.25	376.8	8	159%CPU
swaptions	150.92	292.68	8	193%CPU
x264	64.92	86.98	8	134%CPU
canneal	241.95	222.14	8	92%CPU
dedup	101.09	28.88	8	32%CPU
streamcluster	**268.8**	538.13	8	200%CPU
total	**268.8**	1911.49		