Lawrence Berkeley National Laboratory

Eric Roman

Resilient Runtimes for Global Address Languages

DEGAS Retreat

June 4, 2013

Introduction

Describe resilience efforts in DEGAS project

- BLCR
- Motivation: High-level requirements from a simple performance model
- Design: Role of software components

Project goal

- Deliver a resilient programming environment for PGAS applications

Activities

- Performance models
- Resiliency support for communications layer and language runtime
- Integrated checkpoint/restart
- Programming model for resilience

BLCR Goals

Provide checkpoint/restart for Linux systems running scientific workloads.

Checkpoint and restart shell scripts running MPI applications.

Fit easily into production systems

Run unmodified application source.

Run unmodified binaries. If possible, users should not have to relink codes.

Run on unpatched kernels.

Run with unmodified system libraries. (e.g. libc)

Unrelated features (ptrace, Unix domain sockets) have low implementation priority

Why checkpoint?

We see three main scenarios: scheduling, fault tolerance and debugging.

Example: Migrating A Process

gaius:~ <2>

```
pcp-x-1% ./counting
Counting demo starting with pid 3382
Count = 0
Count = 1
Count = 2
Count = 3
Count = 4
[1]    3382 killed     ./counting
pcp-x-1%
```

xterm

```
n2001% ssh pcp-x-2
Last login: Wed May 14 14:58:12 2008 from old
Have a lot of fun...
pcp-x-2% module load blcr
pcp-x-2% cd /home/pcp1/eroman/src/lbnl_cr/build.pcp-x-1/examples/counting
pcp-x-2% ls
context.3382  counting  counting.o  Makefile
pcp-x-2% cr_restart context.3382
Count = 5
Count = 6
Count = 7

pcp-x-2%
```

Fault Tolerance

Rollback recovery

Not every application can checkpoint itself.

BLCR tries to make every process checkpointable.

Periodic checkpoints

Checkpoint the job at regular intervals.

On system startup, restart jobs from their last complete checkpoint.

Useful for systems with long jobs, fast I/O, and/or high node failure rates.

Status

Processes, process groups and sessions

- Shell scripts (bash, tcsh, python, perl, ruby, ...)
- Multithreaded processes (pthreads with standard NPTL)
- Resources shared between processes are restored.
- Restore PID and parent PID.

Files

- Reopen files during restart: open, truncate, and seek.
- Pipes and named FIFOs
- Files must exist in same location on filesystem
- Memory mapped files are remapped.
- Option to save shared libraries and executable.
- File path relocation

Supported Platforms

Linux kernel 2.6

- test with kernels from kernel.org, Fedora, SuSE, and Ubuntu
- support of custom patched kernels through autoconf

Architectures

- x86, x86-64, ppc, ppc64 and ARM
- Xen dom0 and domU

MPI

- MVAPICH2
- MPICH-V 1.0.x with sockets
- OpenMPI
- Cray Portals
- MPICH2
- SGI

Queue Systems

- Torque support available as of Torque 2.4.
- qhold, qrls, and periodic checkpoints tested.
- BLCR, Condor and Parrot HOWTO available.
- SLURM

MPI

Normal execution with Open MPI

gaius:~ <2>

```
pcp-x-1% !mpir
mpirun -am ft-enable-cr -np 2 lu.A.2


 NAS Parallel Benchmarks 2.2 -- LU Benchmark

 Size:   64x 64x 64
 Iterations: 250
 Number of processes:     2

 Time step    1
 Time step   20
mpirun: killing job...

--------------------------------------------------------------------------
mpirun was unable to cleanly terminate the daemons on the nodes shown
below. Additional manual cleanup may be required - please refer to
the "orte-clean" tool for assistance.
--------------------------------------------------------------------------
pcp-x-1%
```

MPI: Checkpoint/Restart

```
gaius:~ <3>
pcp-x-1% !ps
ps auxw | grep mpirun
eroman    4188  0.2  0.7 114188  3712 pts/0    Sl+  21:17   0:00 mpirun -am ft-e
nable-cr -np 2 lu.A.2
eroman    4196  0.0  0.1   9252   828 pts/3    R+   21:17   0:00 grep mpirun
pcp-x-1% ompi-checkpoint 4188
Snapshot Ref.:   0 ompi_global_snapshot_4188.ckpt
pcp-x-1% kill 4188
pcp-x-1% ompi-restart ompi_global_snapshot_4188.ckpt
 Time step   40
 Time step   60
 Time step   80
 Time step  100
 Time step  120
 Time step  140
 Time step  160
 Time step  180
 Time step  200
 Time step  220
```

Recent BLCR Activity

Queue system support

BLCR, Torque, and OpenMPI

Preemptive scheduling via priority queues under Maui

Incremental checkpointing

Optimizations

(1) **Combining small I/O** requests yields greater I/O efficiency.

(2) **In-kernel compression** of checkpoint data reduces transfer times and storage requirements.

(3) **Incremental checkpointing** reduces transfer times and storage requirements by recording only the state that has changed since the previous checkpoint.

(4) **Memory-exclusion** hints enable user-space code (such as an MPI implementation) to exclude "unimportant" memory from the checkpoint (such as empty receive buffers in an MPI implementation).

(5) **"Live-migration"** moves a still-running process from one compute node to another without need for any intermediate storage.

(6) **In-place rollback** allows the recovery step to return an existing process to state recorded in an earlier checkpoint without the overhead of destroying the process and creating a new one.

Resiliency Modeling Approach

Start from fairly general model, proceed to special cases

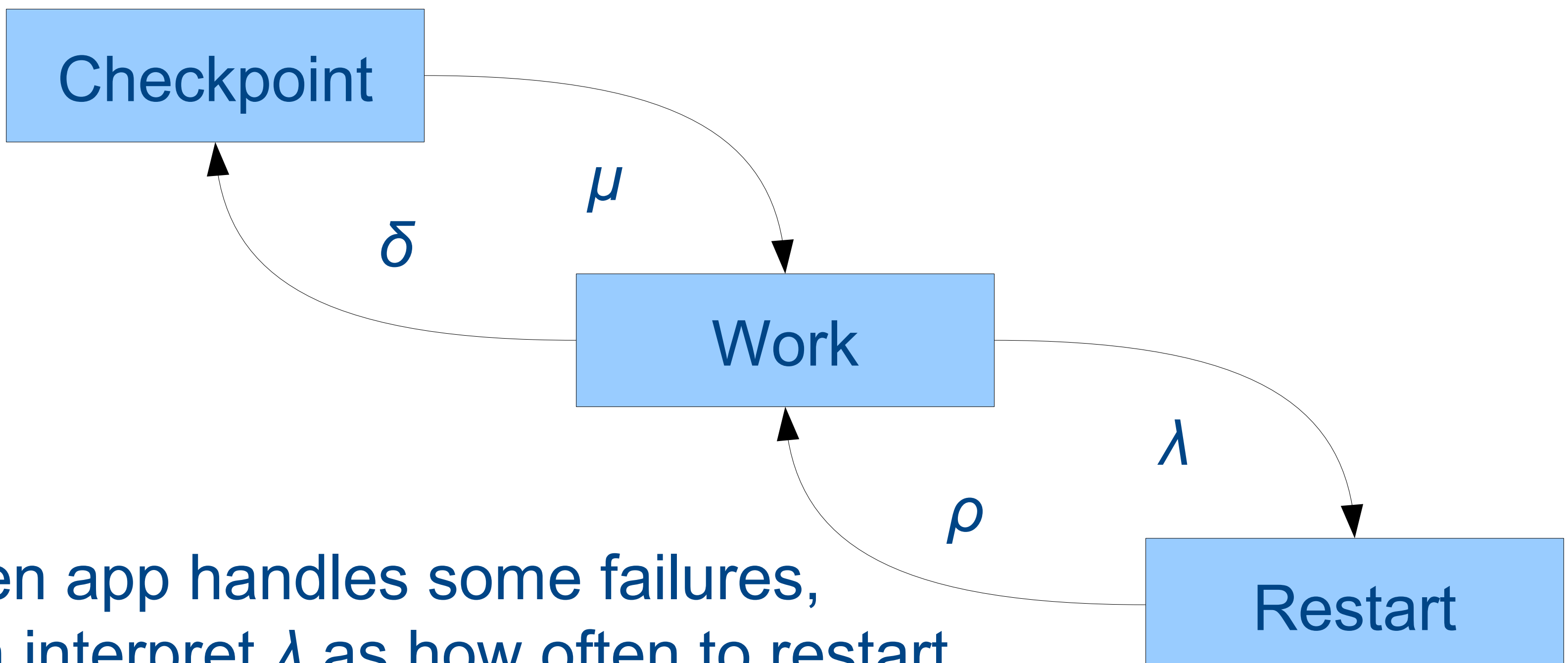
- Construct finite state models of system and express as timed automata
- Approximate timed automata with continuous Markov model

Use methods from reliability engineering to derive performance parameters

	Model	Method
1.	Stochastic timed automata	Discrete event simulation
2.	Markov model	Laplace transform and matrix algebra to solve first-order ODEs
3.	Analytic model	Taylor expansion to first order in failure rate
4.	Algebraic model	Back of an envelope

Three-State Checkpoint Model

Symbol	Rate	Description	Time
λ	Failure rate	Inverse of failure rate	m
δ	Checkpoint rate	Inverse of checkpoint interval	t
μ	Checkpoint speed	Inverse of checkpoint time	c
ρ	Recovery speed	Inverse of restart and rework time	$b + t/2$



When app handles some failures,
then interpret λ as how often to restart

Availability in the Three-State Model

Steady-state probability of being in each state given by vector Π

$$\Pi = \left[\frac{m}{m + b + t/2 + (m/t)c}, \frac{b + t/2}{m + b + t/2 + (m/t)c}, \frac{(m/t)c}{m + b + t/2 + (m/t)c} \right]$$

Working Restart and rework Checkpointing

Optimal checkpoint interval:

Working state probability has an optimum value of t.

$$t_0 = \sqrt{2mc}.$$

$$\Pi = \left[\frac{m}{m + b + \sqrt{2}t_0}, \frac{b + \frac{\sqrt{2}}{2}t}{m + b + \sqrt{2}t_0}, \frac{\frac{\sqrt{2}}{2}t}{m + b + \sqrt{2}t_0} \right]$$

If Checkpoint and Restart Times Equal

1. Assume *c=b*
2. Introduce "dimensionless" checkpoint interval $\tau = t/m = (2c/m)^{1/2}$

$$\Pi = \left[\frac{1}{1+\tau+\tau^2/2}, \frac{\tau/2+\tau^2/2}{1+\tau+\tau^2/2}, \frac{\tau/2}{1+\tau+\tau^2/2}\right]$$

Working — Restart and rework — Checkpointing

To first order:

$$\Pi \approx [1-\tau, \tau/2, \tau/2]$$

In terms of the checkpoint time, we have:

$$\Pi \approx \left[1-\sqrt{\frac{2c}{m}}, \sqrt{\frac{c}{2m}}, \sqrt{\frac{c}{2m}},\right]$$

Optimal Availability by Checkpoint Speed

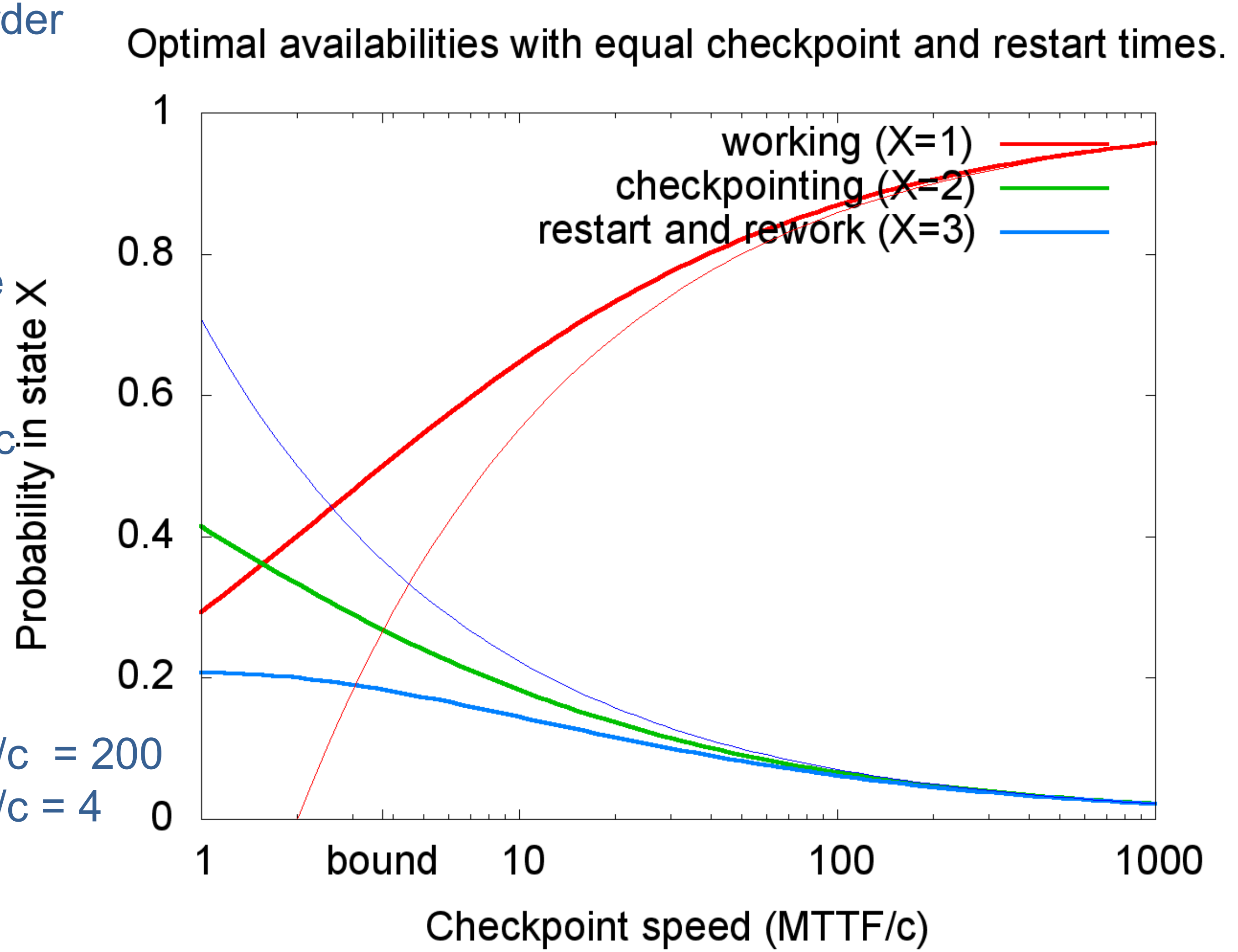
Thin lines show a first-order approximation.

e = (MTTF/c)1/2

Only two parameters are MTTF and c.

100X increase in MTTF/c for 10X decrease in overhead.

10% overhead @ MTTF/c = 200
50% overhead @ MTTF/c = 4



BERKELEY LAB

Overview of Effort

Checkpoint/restart
Containment Domains

BLCR
GASNet
UPC Runtime
Modeling for performance requirements
Load balancing

Two new components:

- Logging
- Replica Management