FFMK: A FAST AND FAULT-TOLERANT MICROKERNEL-BASED SYSTEM FOR EXASCALE COMPUTING

Amnon Barak  
Hebrew University Jerusalem (HUJI)
Hermann Härtig  
TU Dresden, Operating Systems Group (TUDOS)
Wolfgang E. Nagel  
TU Dresden, Center for Information Services and HPC (ZIH)
Alexander Reinefeld  
Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB)

CARSTEN WEINHOLD, TU DRESDEN
ware resources (e.g., cores, caches, memory, and network bandwidth) and the OS/R by preempting at blocking communication calls – a principle called overdecomposition. The units of decomposition are migratable.

To support dynamic management of applications on our platform, we require an application model that is more flexible than the coarse-grained and static division of an application's workload is decoupled from the number execution units or cores. Work that common MPI implementations impose. In our model, the decomposition of an economic model. This economic model will include various aspects such as efficiency, fairness among applications, resiliency, and quality of service. However, its details are still subject to research. Throughput and energy efficiency, fairness among applications, resiliency, and quality of service. The main task of the dynamic platform is to continuously optimize the utilization of the system by means of management: a) Multicore partitions can expand to new colonies, b) Elastic partitions span multiple colonies and shrink, c) Partitions can expand and shrink, d) Elastic applications partitions can expand and shrink.

A redundant set of master nodes monitors and controls the system. The dissemination of the monitoring component (see Section 4.2) and make decisions.

Fig. 3
- **L4 microkernel** controls the node
- **Light-weight** and **low-noise**
- Virtualization: **L⁴Linux** on L4 microkernel
- **Unmodified** Linux programs (MPI, ...)
- **Linux process** = **L4 task** + **L4 threads**
- **Linux syscalls / exceptions**: **generic forwarding** to **L⁴Linux** kernel

---

**L4 Microkernel / Hypervisor**
**Decoupling**: move Linux thread to new L4 thread on its own core
**Decoupling**: move Linux thread to new L4 thread on its own core

**Linux syscall**: Move back to Linux
**DECOUPLED EXECUTION**

- **Decoupling:** move Linux thread to new L4 thread on its own core
- **Linux syscall:** Move back to Linux
- **L4 syscalls:**
  - Scheduling
  - Threads
  - Memory
- **Direct I/O** device access

---

**FFMK: L4 Microkernel + L4Linux as an HPC Operating System**
DECOUPLING: EP

Adam Lackorzynski, Carsten Weinhold, Hermann Härtig, "Decoupled: Low-Effort Noise-Free Execution on Commodity Systems", ROSS 2016, June 2016, Kyoto, Japan

FFMK: Building an Exascale Operating System
Adam Lackorzynski, Carsten Weinhold, Hermann Härtig, "Decoupled: Low-Effort Noise-Free Execution on Commodity Systems", ROSS 2016, June 2016, Kyoto, Japan
NODE ARCHITECTURE

Application

Checkpointing
Platform Management

MPI

InfiniBand, Aries, ...
TCP/IP

Runtime

L4 Microkernel

Linux
Economic model to optimize checkpointing:

- Application lifetime vs checkpoint interval
- Scheduling of checkpoints to reduce conflicts on SSD burst buffers

Online vs offline scheduling examined
FFMK: Building an Exascale Operating System

NODE ARCHITECTURE

Application

Load Balancing

Platform Management

MPI

InfiniBand, Aries, ...

TCP/IP

Linux

Runtime

L4 Microkernel
Balance workload

- **BAD**
  - load=13
  - load=17

- **GOOD**
  - load=12
  - load=12

LOAD BALANCING
- Balance workload
- Minimize communication between partitions
- Balance workload
- Minimize communication between partitions
- Minimize migration
- Balance workload
- Minimize communication between partitions
- Minimize migration
- Compute new partitions fast
Motivation: Diffusive Load Balancing

- Fully distributed method
  - Local operations lead to global convergence
- Practical application is rare
  - Well described since the 1990's
  - Only few papers show real use in HPC

Motivation of this work
- Performance comparison to other state-of-the-art methods at large scale

Schloegel, Karypis, Kumar, SC 2000.

Load per node over iterations
### Performance Comparison: 1Ki-8Ki weak scaling

Max tasks sent + received among all procs:

\[
\text{Max tasks} = \frac{\max(l_v)}{\text{avg}(l_v)} - 1
\]

<table>
<thead>
<tr>
<th>Load imbalance</th>
<th>MigrationMax</th>
<th>EdgeCutMax</th>
<th>Taurus run time (ms)</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2%</td>
<td>500</td>
<td>400</td>
<td>4</td>
<td>300</td>
</tr>
<tr>
<td>4%</td>
<td>1000</td>
<td>800</td>
<td>6</td>
<td>600</td>
</tr>
</tbody>
</table>

Diffusion leads to smallest migration

Diffusion achieves very good edge cut

Diffusion run time ~2 ms for 8192 processes, Zoltan much slower

STEP 1: GOSSIP
Two-stage algorithm: gossip + correction

Main advantage: scalability and resilience (continues to work in presence of failures)

Works for: fault-tolerant broadcast

Next step: extend to operations that include barrier semantics

Future: use in MPI?

Torsten Hoefler, Amnon Barak, Amnon Shiloh and Zvi Drezner, "Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems", Accepted for IPDPS’17, Orlando, FL, USA
Decoupled threads: reduced noise

Checkpointing: Economic model

Diffusion: may be efficient alternative

Corrected Gossip: fault-tolerant broadcast

Work in progress: integrate monitoring + gossip + decision making + migration