Energy Reduction in Cloud and HPC: Design and Implementation of a Blind Methodology

Ghislain Landry TSAFACK CHETSA
Hemera - Avalon Team

Laurent LEFÈVRE, Jean-Marc PIERSON, Patricia STOLF, and Georges Da Costa
High performance computing (HPC)

- User perspective
  - Services that enable new level of innovation and insights
- Technological perspective
  - Cluster of servers, supercomputers, software, tools, interconnects, storage, and services

Example of HPC system.
As the demand for processing grows, HPC is likely to interest all businesses, R&D dept., and academic research

- **Computer aided engineering**: transportation, structural, mechanical design, automotive testing and design
  - Architects use HPC to evaluate building in realistic scenarios through simulation

- **Bio-science and the human genome**: drug discovery, disease control

- **Chemical engineering**: process and molecular design

- **Economics**: a bank uses HPC to analyse high volume of digital transactions
HPC systems and energy consumption

Projections and expectations for exascale

Tianhe-2

K computer

Tianhe-2

Source Top500 list

100MW

+ 65MW

20 MW

12MW

17MW

20 MW

Projections and expectations for exascale
Place great emphasis on a few components: processor architecture, memory subsystems, storage subsystems and communication subsystems

- Performance of cpu-intensive workloads depends on processor architecture

Guarantee performance on average over varied workloads.

- Often result in energy inefficiency
  - Workloads variability and/or variability in a specific workload
Energy reduction in HPC

- **Hardware based initiatives**
  - Multi-configuration (processor, NIC), more energy efficient (GPU), and low power hardware (DDR3, SSD)

- **Software based solutions: application oriented**
  - Rely on multi-configuration hardware
  - Analyse the behaviour or the structure of the application and take energy reduction decisions
  - [Kimura et al. 2010], [Spiliopoulos et al. 2011], [Carrington et al. 2012]
Energy reduction in HPC (cont.)

Software solutions fail to find their way into real HPC deployments

- Complex in nature
  - Vast technical details behind the proposed solution
  - Thorough understanding of the application at hand
- HPC applications are often too complex to be rewritten or modified. “I just added one line and it is no longer working!”
  - limit the scope of application oriented energy reduction techniques
- Often do not reflect current trends

Projection of energy distribution in an exascale system. [Broekema et al. 2012]
Global Objective

A user friendly methodology for reducing the energy consumption of large-scale and distributed infrastructures without a priori knowledge of applications.
1. **A three step methodology**
   - Phase detection
     - Methodology
     - Case study: a real life system
   - Phase/workload characterization
   - Phase identification and system reconfiguration
     - Identification of recurring phases

2. **Experimental results and discussion**
   - MREEF: extension to cloud environments

3. **Conclusions and Perspectives**
Overview

Phase detection
Discover system’s runtime execution patterns

Phase characterization
Associate useful information with known execution patterns

Phase identification and system reconfiguration
Reuse of optimal configuration information for recurring phases
Overview and definitions

Definition:
- Phase: region of execution of the application/system stable with respect to a given metric
- System: computing or storage node

Overview:
- Detect changes in the system’s behaviour that result from changes in the behaviour of programs running
- Two complementary approaches
  - “Power-based phase detection”: make use of the system power profile
  - Execution vector based phase detection
EV-based Phase Detection: system observation

- **Processor and memory activities**
  - Hardware performance counters
    - e.g. `cache_ref` (number of references to the cache), `branch_ins` (number of branch instructions), `cache_misses` (number of cache misses)...
    - Available to all processors, low overhead, used outside of any applications
    - Intrusive (accessed at system runtime)

- **Disk and network usage information**
  - System statistics (`/proc/stat`)
    - Disk read and write counts
    - Network byte send and received counts
EV-based phase detection: execution vector (EV)

**Definition:**
- Column vector whose entries are access rates of sensors – including hardware performance counters, network bytes sent/received and disk read/write counts.
- access per CPU cycles

**Example:**

\[
\begin{pmatrix}
\text{cache_ref} \\
\text{branch_ins} \\
\vdots \\
\text{byteSent}
\end{pmatrix}
\]
Similarity/resemblance between EVs is used for phase detection

- The Manhattan distance between consecutive EVs is the resemblance metric
- Phase changes occur when the distance between consecutive EVs exceeds a given threshold (in percentage of the maximum distance between consecutive EVs).
Phase detection

Step 1

Evolution of the counters access rate for WRF-ARW. The access rate is the number of times a cache line is accessed. The cache ref rate is the number of times a cache line is referenced (hit), and the cache miss rate is the number of times a cache line is not found in the cache (miss).

(c) Cache reference and miss rates along with branch miss rate (WRF-ARW)

(d) Matrix of distances between EVs when running WRF-ARW.

(i,j) distance between EVs sampled at time i and j

Let's see how this works with a real-life workload: the Advance Research Weather Research and Forecasting (WRF-ARW). Figure 5 displays the similarity matrix for WRF, that is the matrix of distances among equivalent values (EVs) when running WRF-ARW.

At point i, j in the matrix the color gives on a gray scale the distance between EVs sampled at time i and j respectively. The darker the color at i, j is the closer EVi and EVj are. Now if we follow the diagonal line which is also the execution time line we can notice 7 triangles, more interestingly in between the triangles distances between EVs are quite big. Consequently using our thresholding algorithm, we can say that WRF breaks into many phases. This is corroborated by the access rate of sensors displayed on figure 4.
Detecting phases of a system running WRF-ARW (Advance Research Weather Research and Forecasting) (2)

(c) Cache reference and miss rates along with branch miss rate (WRF-ARW).
(d) Graphical view of system phase detection.
Contributions: Phase Characterization

- Detect system phase changes
- User friendly
- Has a limited overhead

[GreenCom 2013], [E2DC 2012]
Objective: group phases into classes labelled so that similar phases according to system resource utilization appear under the same label
Six labels: compute-intensive, memory-intensive, mixed, network-transmit, network-received, IO-intensive

- Last level cache references per instruction ratio based (LLCRIR-based) phase characterization
- Statistical-based phase characterization
  - A low overhead phase characterization (Sensor-based)
  - Principal component based phase characterization (PCA-based)
PCA-based phase characterization (1)

Key idea: discover patterns shared by workloads of the same category
Variables have similar pattern w.r.t PC 1 and PC 2 in both cases

(a) Fourier Transform (FT).

(b) Block Tri-diagonal solver (BT)
For each type of phases, variables have similar pattern w.r.t PC 1 and PC 2.

(c) idle_1, variables projected on PC1-PC2 plane.  

(d) idle_2, variables projected on PC1-PC2 plane.
Contributions Phase Characterization

- Low overhead phase/workload characterization techniques
- Adapted to on-line use

[EE-LSDS 2013], [SBAC-PAD 2012]
Key idea: identify recurring phases

- Two main problems
  - Phases are often too long
  - A phase cannot be identified until it has completed

Phase representation

- Use of a single vector: the reference execution vector
  - The closest EV to the centroid of the group of EVs belonging to the corresponding phase
  - Phase identification boils down to comparing reference EVs
Phase identification (2)

Sample results

Graphical view of system phase distributions resulting from five successive executions of *bench_1*.

**Limited number of phases**
**Loosely identify all recurring phases**
Phase prediction

Key idea: reuse of configuration information for recurring phases/workloads

- **Partial phase recognition technique**
  - Identifies an ongoing phase with an existing, before its completion

- **Execution vector classification**
  - Instead of identifying complete phases
    - Match each newly sampled EV with known phases and make the reconfiguration decision for the next sampling interval

*Basic principle:*
- If the system is in a phases labelled say \( l_1 \) at time \( T \) it is likely to be in a phase labelled \( l_1 \) at time \( T + 1 \)
**Def.** Used to refer to any action destined to reduce the energy consumption of the system without “significant performance degradation”

- Performance degradation higher than 10% is significant

**Management practices**
- Platform selection through cross platform energy consumption

**System reconfiguration**
- Memory size scaling
- CPU cores switch on/off
  - Suits best IO and communication intensive workloads
- Traditional power saving schemes
  - Dynamic voltage and frequency scaling (DVFS), adaptive link rate (ALR), disk operating modes
Contributions: Phase Identification and System Reconfiguration

- Design to facilitate on-line use
- Offers alternatives to workload/phase prediction
- Introduce means for reducing the energy consumption of a system

[ICPADS 2012][FGCS 2013]
Outline

1. A three step methodology
   - Phase detection
     - Methodology
     - Case study: a real life system
   - Phase/workload characterization
   - Phase identification and system reconfiguration
     - Identification of recurring phases

2. Experimental results and discussion
   - MREEF: extension to cloud environments

3. Conclusions and Perspectives
MREEF: extension to cloud environments


An implementation of our methodology for reducing the energy consumption of large-scale and distributed infrastructure

Coordinator: system profiling
- implements phase detection; phase characterization;
- phase identification and prediction

Resource utilization metrics

System optimization decisions

Reconfiguration decisions enforcer
- implements power saving schemes;
- captures system resource utilization

Operating system and other tools

- 3000 lines of codes
- Three programming languages:
  - R, Python, and C

MREEF architecture overview.
**MREEF EV classification related results: decentralized (1)**

- Platform size: 34 nodes
- Each node implement the coordinator
- System reconfiguration decisions directed towards the processor through DVFS
  - Processor operating frequency for classes of workloads:
    - Compute intensive: 2.53GHz; mixed: 2.00GHz; memory-intensive: 1.87GHz; all others 1.20GHz
- Test workloads: CG and MG (9 nodes), GeneHunter and POP X1 (4 nodes), and WRF-ARW and MDS (25 nodes); they simultaneously share the 34 nodes
- Use of a majority-rule-based characterization algorithm
  - PCA-based, LLCRIR-based, and Sensor-based phase characterization schemes
Comparison of MREEF (execution time and energy consumption) with the baseline on-demand configuration.

**Energy reduction of up to 15% with less than 7% performance degradation**
MREEF in cloud: context

- Virtual machines (VMs) often have contradictory needs
- Data centres operators often lack information regarding deployed VMs
  - Can be because of privacy concern
- HPC in cloud implies more variability in workloads
MREEF in cloud: experimental protocol and set-up

- 8 VMs deployed on a node having 8 CPU cores
- Each VM executes the following workloads 20 times
  - Cloud workloads:
    - Transactional database system (sysbench + MySQL)
    - Web application (siege + Apache HTTP server)
    - IO intensive application (IOzone)
  - HPC workloads: CG
- Three system configurations
  - on-demand, powersave, MREEF
  - Workloads are the same for each system configuration (a random execution order is provided to each VM)
MREEF in cloud: results

- MREEF configuration:
  - Majority-rule-based phase characterization for phase characterization
  - EV classification for workload prediction

MREEF reduces the energy consumption of up to 8%
Outperforms powersave

Baseline on demand versus powersave and MREEF in a cloud environment.
Conclusions

• A “user friendly” methodology for reducing the energy consumption of large-scale and distributed infrastructures
  • Phase detection, phase characterization and phase identification, and system reconfiguration
  • Allow users to implement their own power saving schemes
  • Takes into account all HPC subsystems and does not require any knowledge of applications from users

• MREEF: a software framework implementing the methodology
  • Energy improvement of up to 15 % without a priori knowledge of application and with less than 7% performance degradation
  • Decentralize and scalable architecture

• Extension to cloud shows that the methodology is not limited to HPC systems
Future directions

- More experiments and developments still need to be conducted in cloud environments
  - Additional power saving schemes: virtual machines migration and/or consolidation
- Investigate cases where we don’t know the application, but its start and end time
  - Can help evaluating our methodology in production environments
- Investigate support for Memory size scaling in future operating systems