Process filtering based on ranking

[1]auth, [2]auth, [3]auth
Department of Computer and Software Engineering
Polytechnique Montreal
Montreal, Canada
{ [1]auth, [2]auth, [3]auth }@polymtl.ca

***Abstract*—**

***Index Terms*—Ranking, Filtering, PageRank, Machine learning, Data mining, Time series, Linux Tracing**

I. Introduction

Tracing is used for performance and bug analysis (importance of tracing in system analysis to detect different kinds of problems)

challenges: trace size is big and needs some heuristics to cope with. Too many data is involved including many nodes,processes, cpus, files, etc. users are only interested in some of them and having all in the analysis can be time consuming and useless.

There are already some techniques in trace collection (sampling or event filtering) or trace analysis (data abstraction) or trace visualization (view time filtering). Sampling filters out some data which might be needed later on abstraction helps to reason about the system in higher levels and may remove some important details visualization: still needs to cope with a hue size of trace and perform analysis on all of them.

In this paper: we perform some early filtering by extracting and highlighting significant entities (e.g., nodes, vms, processes, threads, files, etc.). to do so, we use page rank algorithm to rank entities based on predefined grouping criteria. Graph building and ranking, .

contributions of this work: employ process clustring to group processes which interact in the system together to serve user requests we use page rank algorithm to rank the processes and extract group of important processes we tested and evaluated our method in a real data set collected with lttng kernel tracer.

II. Related Work

tracing is important (cite) different techniques to cope with the size problem abstraction, sampling, etc.

III. Methodology

A. Data collection and pre-processing

Tracing

B. Modeling process interaction as a graph

critical path

Adjacency matrix (or interaction graph creation)

C. Graph community extraction

Once the weighted graph is created, a graph community detection algorithm is applied to partition nodes into distinct modules. Here, we use the Louvain algorithm [1] which has been widely used in many applications to identify the community structures in graphs []. This algorithm has the advantages of rapid convergence, high modularity, and hierarchical partitioning.

The Louvain method consists of two phases. In the first phase, initially, all vertices of the graph are put in different communities. Then, the algorithm sequentially sweeps over all vertices and moves them to one of their neighbors based on the gain in the modularity cost function. The moularity cost function Q can be defined as

$$Q = \sum_{i,j=1}^{N} \left(W_{i,j} - \gamma \frac{k_i^{out} k_j^{in}}{m} \right) \delta\left(c_i, c_j\right) \tag{1}$$

where W is the adjacency matrix, N is the total number of nodes, and m shows the sum of the weight of all the edges in the graph. $k_i^{in/out}$ represents the incoming/outgoing strength of node i. γ is a resolution parameter. $\delta\left(c_i, c_j\right) = 1$ if the community of node i and j are equal ($c_i = c_j$) and $\delta\left(c_i, c_j\right) = 0$ otherwise.

For each vertex i, the modularity Q is computed when putting vertex i in the community of any neighbor j. Then, the community of the neighbor j is picked if it produces the largest increase of Q. The gain of moving the node i to a community c is computed as:

$$\Delta Q_{i \to c} = \sum_{j \in c} \left(W_{ij} + W_{ji}\right) - \frac{k_i^{out} s_c^{in}}{m} - \frac{k_i^{in} s_c^{out}}{m} \tag{2}$$

where $s^{in/out}$ is the sum of the outgoing/incoming strength of nodes in community c.

This iterative procedure continues until no movement yields a gain. At the end of the first phase, the Louvain algorithm obtains the first level partitions.

In the second phase, these communities become supervertices by aggregating each community into a single node. The algorithm reconstructs the graph by aggregating the weight of edges between all supervertices. Two supervertices are connected if there is at least one edge between nodes of the

corresponding communities, in which case the weight of the edge between the two supervertices is the sum of the weights from all edges between their corresponding partitions at the lower level.

These two phases are then recursively applied to the supergraphs in a hierarchical way. The algorithm terminates until communities become stable. Typically, the Louvain algorithm converges very quickly and can identify communities in a few iterations.

D. Ranking processes

In this step, each community is considered as a separate graph and processes within each partition are ranked. The PageRank (PR) algorithm [2] is employed to assign a rank to each process. The PR approach has been used by the Google search engine to model the importance of Web pages. This algorithm counts the number of links to a Web page to evaluate the importance of that page. The idea is that highly reputable Web pages are more likely to be cited by other reputable documents. Here, the same idea is applied to rank the processes that are more significant based on interactions with other processes.

Let p and v be two processes in community c and E_c the set of directed edges in c. The PageRank of p, $PR_j(p)$, is defined using the following recursive formula:

$$PR_t(p) = \alpha \sum_{(p,v)\in E_c} \frac{PR_{t-1}(v)}{out - degree(v)} \tag{3}$$

where α is a normalization factor for the total rank of all processes and t shows the iteration number. The PageRanks are initialized as $\frac{1}{n}$ for each process, where n is the number of nodes in each community. The PageRank values get updated in each iteration, until it converges into a stable value.

IV. Use cases

A. Use case I

B. Use case II

Cloud computing provides on-demand access to massive amount of shared resources. Despite the flexibility brought by this infrastructure, the complexity of monitoring such environment is high. One effective approach to reduce this complexity is to take advantage of VM clustering techniques. These approaches cluster VMs with similar behavior together. Thus, the service administrator deals with a group of clusters rather than thousands of individual VMs. This facilitates the workload monitoring and resource management procedure.

In this work, different monitoring metrics and features are extracted from each VM to perform clustering in a cloud environment. Here, the metrics are extracted from host hypervisor in an agent-less manner without having an internal access to the VMs. A feature extraction strategy is applied to produce a more compact data representation.

TABLE I
THE EXTRACTED METRICS FROM THE STREAMING TRACING LOG OF EACH VM

Metric	Description
$vCPU_{Root}$	Root mode average time
$vCPU_{non-Root}$	Non-Root mode average time
f_{Disk}	The frequency of wait for Disk
f_{Net}	The frequency of wait for Net
f_{Timer}	The frequency of wait for Timer
f_{Task}	The frequency of wait for Task

C. Workload based feature extraction

1) Trace data collection: The workload on each VM is traced using the Linux Trace Toolkit Next Generation (LTTng) [3], which provides low overhead kernel and userspace tracing. In addition, the Kernel Virtual Machine (KVM) [4] is used as our hypervisor virtualization which is compatible with LTTng toolkit.

2) Workload related feature extraction: The collected tracing data are fed to the Tracecompass [5] open source tool, which includes several pre-built modules to view and analyze traces. In this work, the agent-less VM analysis module is used to collect useful information from streaming traces and stores it into a State History Tree (SHT) database. Monitoring metrics are then extracted from the State History Tree by querying the SHT attributes. Based on our domain knowledge, we collect different metrics per VM as listed in Table I.

The $vCPU_X$ metric with $X \in \{Root, non - Root\}$ shows the average time that a VM is running in a VMX root or VMX non-root mode. VMX (or Virtual Machine Extension) is a set of instructions on x86 processors that support virtual interrupt injection into VMs. Two types of VMX operation exist: **VMX root** and **VMX non-root**. A vCPU on a VM executes the non-privileged instructions in a non-root mode while runs the privileged instructions as root mode.

Other sets of features considered in our analysis are based on different interrupts that causes the host scheduler to wake up the vCPU thread. The interrupts could be divided in to the following categories: **(I)** Timer interrupt, a process inside the VM sets a timer and the timer is fired; **b)** Task interrupt, inter processor communication wakes up a process; **c)** Network interrupt, a process inside the VM waits for an incoming packet from a remote process; **d)** Disk interrupt, a process inside the VM waits for disk. The f_X metrics with $X \in \{Disk, Net, Timer, Task\}$ shows interrupt related features in Table I. These features represent the waking up frequency of a vCPU by different interrupt signals.

3) Process ranking and VM clustering: The proposed filtering approach is applied on each VM to extract the most significant processes. Then, the features listed in Table I are extracted only from the top-ranked processes. In the next step, the performance metrics of the top important processes are fed into a two phase clustering approach based on K-Means. Inter-clustering and intra-clustering similarity metrics of the silhouette score is used to depict the distinct workload groups.

V. Evaluation

the overhead of grouping and highlighting Importance of the filtering

VI. Conclusions

In this paper,

References

[1] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, oct 2008.

[2] W. Xing and A. Ghorbani. Weighted pagerank algorithm. In *Proceedings. Second Annual Conference on Communication Networks and Services Research, 2004.*, pages 305–314, May 2004.

[3] Mathieu Desnoyers and Michel R. Dagenais. The lttng tracer: A low impact performance and behavior monitor for gnu/linux. In *OLS (Ottawa Linux Symposium) 2006*, pages 209–224, 2006.

[4] (kvm) kernel virtual machine. http://www.linux-kvm.org/page/Main_Page. Accessed: 2019-01-30.

[5] Tracecompass. https://projects.eclipse.org/projects/tools.tracecompass. Accessed: 2019-01-30.