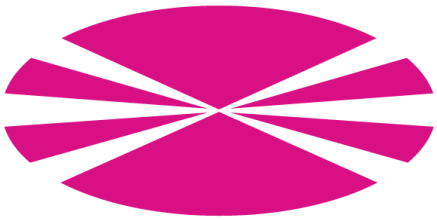

University of A Coruña

Computer Architecture Group

BDWatchdog User Guide

Authors:
Jonatan Enes, Roberto R. Expósito,
Juan Touriño

July 29, 2019

1 Overview

BDWatchdog is a framework to assist in the in-depth and real-time analysis of the execution of Big Data frameworks and applications. Two approaches are used in order to get an accurate picture of what an application is doing with the resources it has available 1) per-process resource monitoring and 2) mixed system and JVM profiling using flame graphs.

With monitoring and profiling, used individually or combined, it is also possible to easily identify both resources and code bottlenecks as well as account for resource utilization or spot certain patterns that frameworks or applications may have.

In addition, integrated in BDWatchdog, a tool has been designed and implemented to create a serverless container infrastructure where the resource limits (i.e., CPU, Memory, disks, network) of such containers can be rescaled in real time according to multiple user-defined policies such as the resource usage of the applications running inside a container or any desired time-varying resource limit. This tool has also been used in a specific scenario to set a limit on the energy consumption of a container or set of containers by using CPU scaling properly.

1.1 Monitoring

For the per-process monitoring, resource usage metrics for CPU, memory, disk and network are retrieved, processed and pushed to a time series database in a continuous stream. Thanks to this stream-based approach, it is possible, in real-time, to see and use this data for several purposes like visualization, reporting or to take automated actions to tackle several situations like resource bottlenecks, loss of efficiency or even resource limitation.

Moreover, the per-process approach allows to deploy this monitoring solution on a broader type of virtualized infrastructures, from the already common bare-metal hosts or virtual machines (Cloud instances) to the newer type of containers.

Finally, thanks to the use of time series as the means to store the data, it is possible to perform several operations through filtering and aggregation, which may overall provide richer reports (e.g., Show the aggregated CPU usage of a command or process across a cluster, show the average disk bandwidth of several hosts).

1.2 Profiling

For the real-time profiling of Java-based applications, the perf utils are used to continuously get system-wide CPU stacks, using a configurable frequency. The stacks are then processed, marked to differentiate them across applications and pushed to a document-based database. Once in the database, the stacks can be retrieved and processed with several tools (console script, web interface) to create the interactive flame graphs.

These flame graphs can be created in real time with data collected between two time points on an application's execution. Once created, information regarding the percentage of spent time for each class can be interactively analyzed in a recursive way by following the stack calls.

1.3 Resource autoscaling

The resource autoscaling tool implemented as part of BDWatchdog allows to create a serverless environment which offers several interesting features that can be exploited from both the user and the provider point of view:

- Increased resource utilization, as it is possible to dynamically adjust the resource scheduling to match the real resource usage of the containers.
- A serverless environment is provided and thus, only the used resource amounts are acccounted for, being the provider's task to adjust the resources provided as needed while minimizing any performance impact.
- Increased serverless flexibility is offered to the user as by using software containers, which offer a similar environment to virtual machines, a broad range of applications can be deployed and executed.

1.3.1 Energy monitoring and limit enforcement

The energy monitoring and capping features are created by using both the *BDWatchdog* monitoring capabilities as well as the *Autoscaling tool*. This combination makes it possible to rescale the CPU in real time and thus adjust the energy that a container or a set of containers consume at any moment.

With this energy management some interesting features can be exploited:

- Consider energy as another accountable resource and thus make it possible to be shared and accounted between the users/containers.
- Apply an energy limit to a container(s)/user and have it enforced at any moment.

2 Modules, Requirements and Installation

2.1 Resource metrics generator

This module serves the purpose of generating the time series that are later used for the monitoring capabilities.

To be used this module requires an *OpenTSDB* database and the *atop* (for CPU, memory and disk resources) and *nethogs* (for network) programs. Scripts are provided for the installation of these programs on CentOS and on debian-based Linux distributions.

The source code can be found on `https://github.com/JonatanEnes/bdwatchdog/tree/master/MetricsFeeder`.

2.2 Flame graphs generator

This modules generates the flame graph data that can be later used to create the flame graphs for the profiling capabilities.

To be used this module requires access to the Linux *perf* command as well as a *MongoDB* database and *EVE* as a REST API.

The source code can be found on `https://github.com/JonatanEnes/bdwatchdog/tree/master/FlamegraphsGenerator`.

2.3 Web user interface

This module is a web interface that can be used to visualize the time series that represent the monitoring and the flame graphs for the profiling. This user interface has been programmed entirely with HTML + JS + CSS and acts as a static front end that pulls data from the same databases used to push the data in the generators.

No specific installation or requirement is needed other than downloading the code and accessing the web (index.html file) with a browser.

The source code can be found on `https://github.com/JonatanEnes/bdwatchdog/tree/master/MetricsFeeder`.

2.4 Experiment timestamp management tool

This module is used to manage timestamps for an experiment run and its tests executions. These timestamps are stored in the form of JSON documents with metadata of the experiments and tests including the starting and ending UNIX timestamps. These timestamps are later used by other modules such as the Web user interface to plot monitoring or profiling specifically between two time points.

To use this module a *MongoDB* database is needed to store the JSON documents as well as *EVE* as a REST API.

The source code can be found on `https://github.com/JonatanEnes/bdwatchdog/tree/master/TimestampsSnitch`.

2.5 Automatic resource scaler

This module implements the automatic resource scaling of resources such as CPU, memory, disk and network.

To properly deploy this tool, a *CouchDB* database is needed. In addition, if this tool is used to use its energy capping capabilities, an external program capable of producing container-level energy metrics is needed, currently only *PowerAPI* has been integrated and is supported.

The source code of this tool can be found on `https://github.com/JonatanEnes/AutomaticRescaler`.

The webpage of PowerAPI is `https://powerapi.readthedocs.io/en/latest/intro.html`.

3 Usage

BDWatchdog has been designed and implemented in a modular way and thus, all of its features can be used combined or independently.

For monitoring and profiling, the recommended way of using BDWatchdog is to deploy the *Resource metrics* and *Flame graphs generators*, execute any task or experiment and mark the start and ending times with the *Ex-*

periment timestamp management tool and finally plot the results using the *Web user interface*.

For the resource autoscaling and energy capping functionalities, the monitoring capabilities of BDWatchdog must be deployed as well as the services of the *Automatic resource scaler*. Once deployed there are several scripts and configurations value to modulate the behaviour of this tool.

For more information regarding the usage of BDWatchdog please consult its user guide at `https://bdwatchdog.readthedocs.io/en/latest/`.

4 Contact

BDWatchdog has been entirely developed in the Computer Architecture Group at the University of A Coruña by the following authors:

- Jonatan Enes Alvarez `http://gac.udc.es/~jonatan.enes/`
- Roberto R. Expósito: `http:gac.udc.es/~rreye`
- Juan Touriño: `http:gac.udc.es/~juan`

For any question regarding BDWatchdog, whether about the source code, its usage or functionalities, please address via email to Jonatan enes (jonatan.enes@udc.es).