Demo: Emulating Distributed Workloads with DistWalk

Remo Andreoli, Tommaso Burlon, Antonio Napolitano, Tommaso Cucinotta
Scuola Superiore Sant'Anna, Pisa, Italy
name.surname@santannapisa.it

***Abstract*—This demo showcases DistWalk [2], an open-source distributed workload emulator designed to study the end-to-end latency implications of Linux-based systems. DistWalk is capable of deploying sequences of compute, network, and storage operations arranged within graph-like topologies, to be carried out across multiple servers. It supports a variety of communication protocols and traffic patterns, and enables the customization of several factors, such as the duration and parallelism of compute-intensive operations, the network security and connection handling strategy, and the I/O data access and synchronization mode, among others. DistWalk can be used to experiment with a variety of deployment models for distributed workloads, from bare-metal to virtualized or containerized environments, e.g., using Cloud/Edge infrastructures, OpenStack, Kubernetes, or other orchestrators. This allows Cloud/Edge researchers and developers to perform experimental comparisons of the latency achievable by distributed workload patterns across a wide range of system-level configurations.**

***Index Terms*—Workload Emulation, Distributed Systems, Cloud Computing, End-to-End Performance**

I. Introduction and Motivations

When engineering distributed or Cloud-native services, it is useful to assess the expected achievable performance early in the software life-cycle, when the implementation may not be (completely) available yet. This is particularly useful for time-critical applications [1], where the experienced end-to-end latency is heavily influenced by design choices such as the application parallelism degree, the placement of its components throughout a Cloud/Edge infrastructure, the workload load-balancing strategy, the use of secure protocols and the configuration of the available tunables for the machines, software stack, in-kernel scheduler, or other system-level parameters.

In order to estimate the performance implications of the available design choices and platform tuning options, it is commonplace to resort to experimental approaches. This may be conveniently performed by using computational, networking and disk I/O *microbenchmarks*.

For example, `stress-ng` is designed to stress various components of a computing system: the capabilities of the physical CPU, cache, and the whole memory hierarchy, the I/O subsystem, various features of the OS kernel, from page swapping to inter-process communications through pipes or local networking, to high-frequency timers, and others.

Another example is `fio`, a tool dedicated to testing the I/O subsystem of a platform, with the ability of emulating a variety of I/O workloads, including parallel scenarios with several threads requesting highly heterogeneous types of I/O operations (random vs sequential access patterns, direct vs buffered operations, etc...).

For the network, a well-known micro-benchmark is `iperf`, a tool designed to measure the maximum achievable end-to-end bandwidth between a client and a server communicating over IP networks (either with TCP or UDP).

However, these tools usually test a single component or aspect of a distributed system (e.g., the computational, disk I/O, or networking performance of a single component, link, or communication path). As a result, they provide limited insight into how a distributed application with geo-dislocated components might perform under certain conditions.

The DistWalk tool [2] that will be presented in this demo, allows for: i) exploring several low-level optimization opportunities in the design space and deployment environment; and ii) verifying experimentally the impact of these design choices, and associated tunables, on the end-to-end performance that might be expected by the final software.

II. DistWalk Architecture

DistWalk [2] is an open-source distributed workload emulator, with the ability to measure the resulting end-to-end latency for a variety of resource consumption patterns, providing a wide range of low-level options to mimic common designs and operating conditions exhibited by a distributed workload. Its main features are: i) a highly scalable architecture featuring asynchronous, socket-based communications; ii) low-level tunables for fine-grained control over the workload behaviors (i.e., data access pattern, compute processing duration, etc.) and server configuration details (i.e., communication model, data synchronization, etc); iii) ready-to-use executables that can be integrated, in part or entirely, within or alongside other third-party tools; and iv) a platform-agnostic tool that can be deployed on-premise, on small-scale clusters, or on cloud platforms, using orchestrators like OpenStack or Kubernetes.

DistWalk follows a client-server architecture. The simplest scenario involves a *client* that defines a sequence of operations, and a *server* (referred to as a *node* in DistWalk) that executes them. The sequence is encapsulated in a network packet and sent to the node multiple times, with inter-request times distributed according to an initial request rate and a series of rate ramp-ups. For each request, the server performs the user-defined operations, then it sends a response to the client. The client awaits for responses to all requests, optionally up to a maximum timeout, before displaying the measured round-trip latencies, a.k.a., response times, for each request. More interesting scenarios involve multiple nodes and a client submitting workload structured as a graph-alike topology that spans across multiple nodes. The workload is specified

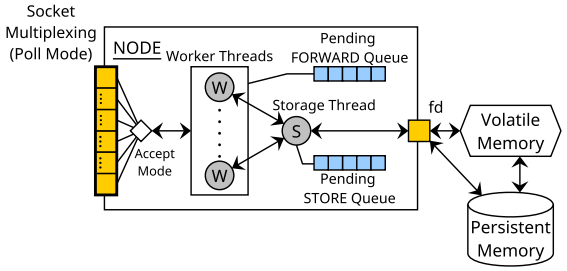

Fig. 1: Architecture of a DistWalk node: each component offers a high degree of configurability (from [2]).

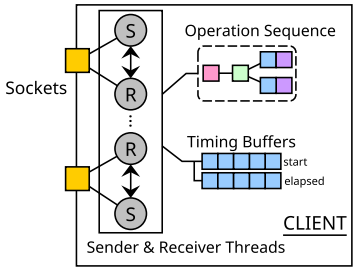

Fig. 2: Architecture of a DistWalk client (from [2]).

using a rich set of command-line options, that can also be arranged in a script-alike program [2]. For each request, the server performs the enclosed operations, then forwards the remaining payload to 1 or more servers, waiting for all or a configurable number of replies, e.g., waiting for a majority of them is useful to emulate quorum-based replication protocols. Additionally, replies that are not received within an optional configurable timeout are retried up to a configurable number of times [3]. Once the payload is fully exhausted, a response is routed back to the client, following the communication path in reverse order. All per-request parameters, from request inter-arrival times, to processing times on the nodes, to the data size of request and response messages, and data to be written to or loaded from disk, can be specified in terms of fixed values, or values drawn from common probabilistic distributions with specified parameters, or values read from a CSV file column. Finally, multiple clients may be used to submit various workloads concurrently onto the same nodes.

The client and node components are multi-threaded C programs with no dependencies on external libraries, with the exception of the OpenSSL crypto and SSL libraries, if SSL is enabled. This design ensures that DistWalk is a high-performance, highly scalable, plug-and-play toolkit. Both components are controlled via the command line using a versatile syntax. The DistWalk node is a networked server designed to be reused across several client workload runs. Figure 1 depicts the major features of the node architecture: i) the ability to choose between different connection accept policies (called "accept modes" in DistWalk) and multiplexing mechanisms (called "poll modes"), and the optional use of TLS/SSL if secure communications are needed; ii) a pool of threads to handle the user-defined operations, both synchronously and asynchronously; and iii) a dedicated thread for interacting with persistent memory. Figure 2 illustrates the client architecture, featuring pairs of sender and receiver threads to interact with the nodes. This two-threads architecture has the advantage of keeping the request submission period/rate as stable and precise as possible.

DistWalk has been used for the experimental evaluation of a number of prior research papers [4], [5], [7]–[9], [11], but only recently a comprehensive overview of its design, features and command-line options has been published in [2].

III. Demo Content

The presenter will walk through a subset of examples adapted from [2], but reinterpreted from a single-system perspective. Participants who wish to follow along must have access to a Linux-based system, and Internet access to download DistWalk[1]. The README included within the DistWalk package contains the step-by-step instructions for setting up and running an experiment.

This demo will focus on studying latency implications of two essential aspects of cloud-based services: quorum-based replication [10] and load balancing [6].

References

[1] R. Andreoli, R. Mini, P. Skarin, H. Gustafsson, J. Harmatos, L. Abeni, and T. Cucinotta. A multi-domain survey on time-criticality in cloud computing. *IEEE Transactions on Services Computing*, page 1–19, 2025.

[2] Remo Andreoli and Tommaso Cucinotta. DistWalk: a Distributed Workload Emulator. In *25th IEEE international Symposium on Cluster, Cloud and Internet Computing*, Tromsø, Norway, May 2025.

[3] Tommaso Burlon. Adding multiple features to the emulation tool of distributed systems distwalk, Sant'Anna School of Advanced Studies. https://dta.santannapisa.it/t/etd-10162023-221502/, December 2023.

[4] Tommaso Cucinotta, Luca Abeni, Mauro Marinoni, Alessio Balsini, and Carlo Vitucci. Reducing temporal interference in private clouds through real-time containers. In *IEEE International Conference on Edge Computing*, pages 124–131, 2019.

[5] Tommaso Cucinotta, Luca Abeni, Mauro Marinoni, Riccardo Mancini, and Carlo Vitucci. Strong temporal isolation among containers in OpenStack for NFV services. *IEEE Trans. on Cloud Computing*, 2021.

[6] Pawan Kumar and Rakesh Kumar. Issues and challenges of load balancing techniques in cloud computing: A survey. *ACM computing surveys (CSUR)*, 51(6):1–35, 2019.

[7] Giacomo Lanciano, Remo Andreoli, Tommaso Cucinotta, Davide Bacciu, and Andrea Passarella. A 2-phase strategy for intelligent cloud operations. *IEEE Access*, pages 1–1, 2023.

[8] Giacomo Lanciano, Filippo Galli, Tommaso Cucinotta, Davide Bacciu, and Andrea Passarella. Predictive auto-scaling with OpenStack Monasca. In *14th IEEE/ACM Intl. Conf. on Utility and Cloud Comp.*, Dec. 2021.

[9] Giacomo Lanciano, Filippo Galli, Tommaso Cucinotta, Davide Bacciu, and Andrea Passarella. Extending openstack monasca for predictive elasticity control. *Big Data Mining and Analytics*, 7(2), June 2024.

[10] Saif Ur Rehman Malik, Samee U Khan, Sam J Ewen, Nikos Tziritas, Joanna Kolodziej, Albert Y Zomaya, Sajjad A Madani, Nasro Min-Allah, Lizhe Wang, Cheng-Zhong Xu, et al. Performance analysis of data intensive cloud systems based on data management and replication: a survey. *Distributed and Parallel Databases*, 34:179–215, 2016.

[11] Riccardo Mancini, Tommaso Cucinotta, and Luca Abeni. Performance modeling in predictable cloud computing. In *10th International Conference on Cloud Computing and Services Science - CLOSER*, pages 69–78. INSTICC, SciTePress, 2020.

[1] See: https://github.com/tomcucinotta/distwalk