

## A. Artifact Appendix

### A.1 Abstract

This publicly available artifact contains a snap-shot of Hermes(KV) repository. The repository consists of the Hermes protocol specification (e.g., for model checking) and all the code required to reproduce the experiments of Hermes from the ASPLOS’20 paper “Hermes: Fast and Reliable Data Replication with Linearizability”.

The provided artifact supports this work in two ways:

- *Experimental evaluation* of Hermes(KV) performance.
- *Correctness evaluation* of Hermes via TLA+ model checking.

Requirements for the *experimental evaluation*: root access to a cluster of (i.e., 2 or more) nodes interconnected through Infiniband RDMA network cards and switches. Although experimental evaluation is sensitive to the environment and the chosen hardware, we expect the trends as presented in the paper to hold for similar configurations (e.g., same number of nodes and network bandwidth).

There are no specific requirements for the correctness evaluation (other than Java 1.8 environment support). The expected result is successful model check termination without errors.

### A.2 Artifact check-list (meta-information) – *Experimental*

- **Note:** Sections between §A.2–§A.7 solely discuss *experimental evaluation*, while subsequent sections detail *correctness evaluation*. Thus, check-list for *correctness evaluation* is on §A.8.
- **Algorithm:** Hermes reliable replication protocol.
- **Program:** HermesKV
- **Compilation:** Gcc with C++11 support (public 7.4.0 – tested); detailed instructions to compile and run Hermes(KV) are given below.
- **Binary:** The artifact includes Makefiles to build the “hermesKV” exec.
- **Run-time environment:** Linux OS (tested on Ubuntu 18.04 4.15.0-55-generic) with root access and bash (4.4 tested); Mellanox OFED (ibverbs) drivers; (lib)memcached and (lib)numa libraries.
- **Hardware:** A homogeneous cluster of x86-64 nodes interconnected via (publicly available) Infiniband RDMA networks cards and switches.
- **Run-time state:** Sensitive to network/cache contention.
- **Execution:** Sole-user, thread pinning, huge-page pinning.
- **Metrics:** Throughput (M.Requests/sec) and request latency (in  $\mu$ s).
- **Output:** Numerical results in console output and dumped text files.
- **Experiments:** Manual steps by user; results may vary based on the hardware setup and threads used to reproduce the experiment.
- **How much disk space required (approximately)?:** <15 MB (without skewed traces); 450MB (with skewed traces included).
- **How much time is needed to prepare workflow (approximately)?:** ~1 hour.
- **How much time is needed to complete experiments (approximately)?:** 1-3 min / data point (~8 human hours in total).
- **Publicly available?:** Yes
- **Code licenses (if publicly available)?:** Apache 2.0
- **Archived (provide DOI)?:** <http://doi.org/10.5281/zenodo.3563199>

### A.3 Description

#### A.3.1 How delivered

The artifact is publicly available through the following DOI:

<http://doi.org/10.5281/zenodo.3563199>.

The latest version is available on the git repository:

<https://github.com/akatsarakis/Hermes>.

All the artifact files are licensed under the Apache 2.0 license.

#### A.3.2 Hardware dependencies

The experimental evaluation of Hermes(KV) requires a cluster of x86-64 nodes interconnected via Infiniband RDMA networks cards and switches (tested on Mellanox “ConnectX-4” equipment).

#### A.3.3 Software dependencies

The Hermes(KV) code is developed for *Linux* OS (tested on Ubuntu 18.04 4.15.0-55-generic) and requires root access. *Mellanox OFED* ibverbs drivers (4.4-2.0.7.0 tested) must be installed for RDMA networking. The *libmemcached* library is used for setting up the RDMA connections and *libnuma* for binding huge-pages.

#### A.3.4 Data sets

Traces of read/write with configurable write ratio and the following data accesses patterns: 1. uniform (internal to hermesKV) 2. skewed (given as input – enabled via `/hermes/include/config.h`).

### A.4 Installation

The instructions below call a particular node from the cluster as the *manager* (pick any but only one) and the rest are the *members*. On every cluster node:

1. **Download the artifact in the user’s home directory.**
2. **Install Mellanox OFED drivers** (tested on 4.4-2.0.7.0) from: [https://www.mellanox.com/page/products\\_dyn?product\\_family=26](https://www.mellanox.com/page/products_dyn?product_family=26)
3. **Run the `/hermes/bin/setup.sh`**; which installs the required libraries and does the network and memory configuration setup. *-Note:* `setup.sh` must run again if a node reboot occurs.

Only on *manager*:

1. **Edit the `/exec/hosts.sh`** to set `ALL_IPS` and `LOCAL_IP` variables that indicate the IP(s) of the RDMA-enabled network cards. The first entry of `ALL_IPS` array must be the manager’s IP followed by the IP addresses of the whole cluster. The `LOCAL_IP` must be dynamically calculated based on the node that invokes the script. Finally set the RDMA device name (i.e., the `hca_id` of the device when executing `ibv_devinfo` command)
2. **Edit the `hermes/include/hermes/config.h` header file** to configure the *setup and default parameters*.
3. From `/hermes/exec` directory **compile *hermesKV*** using `make`.
4. **Copy over the network** (e.g., using `scp`) the *hermesKV* exec. & the edited `hosts.sh` to `/hermes/exec` directory of **all members**.

### A.5 Experiment workflow

1. First on manager and then on all members run the `run-hermesKV.sh` from `/hermes/exec` directory. Use the same cli arguments (i.e., experiment parameters §A.7) for the script in all of the nodes.
2. During the experiment periodically printed console outputs show the per-node throughput in million requests per second.
3. After a predefined number (e.g., four by default) of those console outputs the experiment is terminated and the local throughput (and request latency if enabled – in  $\mu$ s) are dumped to files.

Below is an example of an experiment invocation with three nodes(M), 16 worker threads(W) per-node and 50% of the accesses are writes (w – the argument is divided by 10 to give percentage):

```
manager:~/hermes/exec$ ./run-hermesKV.sh -M 3 -W 16 -w 500
```

```
members:~/hermes/exec$ ./run-hermesKV.sh -M 3 -W 16 -w 500
```

A detailed list of cli arguments that are supported can be found within the `run-hermesKV.sh` script. Note that if `hosts.sh` includes more than three nodes the additional nodes will eagerly terminate.

## A.6 Evaluation and expected result

The evaluation consists of numerous experiments that vary the script parameters of run-hermesKV.sh. The experimental procedure for rCRAQ is similar but uses run-rCRAQ.sh instead.

These performance experiments are sensitive to resource contention and the environment parameters such as the number of nodes, threads and the available network and PCIe bandwidth. Nevertheless, we expect the general trends and the Hermes advantage as shown in our evaluation to be apparent in similar configurations.

## A.7 Notes

**Experiment parameters.** The experiment parameters that can be given as cli arguments to the run-hermesKV.sh script are detailed inside the script. Those arguments are optional and if they are not specified the default value are set based on the /hermes/include/config.h.

**Experiment automation.** The /hermes/bin directory contains some useful scripts that can be run via the manager node to fully automate the executable sharing, the run of the experimental procedure. However, those scripts require ssh-keys and the GNU parallel command and thus are out of the scope of this brief artifact.

**Latency dump.** If the request latency measurement cli argument is enabled then at the end of execution the manager node dumps a file with latency stats. There is a latency parser in /hermes/bin that can be used to make sense of the result and can be used as follows:

```
manager:/hermes/bin$ ./csv_latency_parser.py < dump_name.csv
```

## A.8 Artifact check-list (meta-information) – Correctness

- **Algorithm:** Hermes reliable replication protocol
- **Binary:** The TLA+ toolbox
- **Run-time environment:** Any OS with Java runtime 1.8 or later.
- **Execution:** No specific conditions during execution; run-time varies according to model constants: 5 mins (w/ 3 nodes & max\_version 2); ~5 hours 20 mins (w/ 3 nodes & max\_version 3)
- **Metrics:** Correctness is binary – (no) errors.
- **Output:** TLA+ Toolbox console output.
- **Experiments:** No variation of results.
- **How much disk space required (approximately)?:** ~1 GB for TLA+ Toolbox; for model checking it varies according to the model constants.
- **How much time is needed to prepare workflow (approximately)?:** 10 minutes.
- **How much time is needed to complete experiments (approximately)?:** See "Execution" above.
- **Publicly available?:** Yes
- **Archived (provide DOI)?:** <http://doi.org/10.5281/zenodo.3563199>

## A.9 Description

To evaluate (i.e., model check) the correctness of Hermes protocol you need to download and install the TLA+ Toolbox so that you can run the TLC model checker using the Hermes TLA+ specification.

### A.9.1 How delivered

See §A.3.1.

### A.9.2 Software dependencies

Any OS with Java 1.8 or later, to accommodate the TLA+ Toolbox.

## A.10 Installation

Download and install "The TLA+ Toolbox" following the guidelines that exist in the link below:

<https://lampport.azurewebsites.net/tla/toolbox.html>

## A.11 Experiment workflow

### 1. Launch the TLA+ Toolbox.

2. **Create a spec:** *File>Open Spec>Add New Spec...*; Browse and use *hermes/tla/HermesRMWs.tla* as root module to finish.

3. **Create a new Model:** Navigate to *TLC Model Checker>New model...*; and create a model with the name "hermes-model".

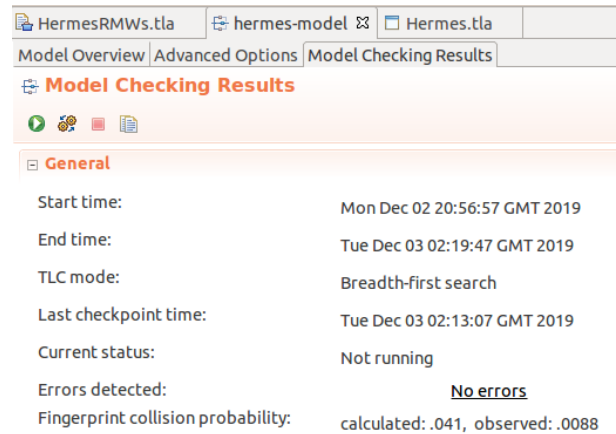
4. **Setup Behavior:** In *Model Overview* tab of the model, and under the "What is the behavior spec?" section, select "Temporal formula" and write "HSpec".

5. **Setup Constants:** Then specify the values of declared constants (under "What is the model?" section). You may use low values for constants to check correctness without exploding the state space (§A.13). An example configuration would be three nodes and a maximum version of two or three (see checklist for approximated run-times). To accomplish that you would need to click on each constant and select the "ordinary assignment" option. Then fill the box for H\_MAX\_VERSION with a small number (e.g., with "2" or "3") and for H\_NODES with a set of nodes (e.g., "{0,1,2}" – for three nodes).

## A.12 Evaluation and expected result

**Run the TLC model** through the window menu *TLC Model Checker>Run model*.

**Expected result:** The TLC model should successfully terminate after exhausting the state space and checking the protocol for deadlocks and correctness invariants in every state. There should be no errors other than running out of disk for big constants (§A.13). Successful termination can be checked by navigating into the hermes-model's *Model Checking Results* tab and looking at the *General* section the fields *Current status* and *Errors detected* must be "Not running" and "No errors" respectively as shown in Figure 1.



HermesRMWs.tla	hermes-model	Hermes.tla
Model Overview	Advanced Options	Model Checking Results
Model Checking Results		
General		
Start time:	Mon Dec 02 20:56:57 GMT 2019	
End time:	Tue Dec 03 02:19:47 GMT 2019	
TLC mode:	Breadth-first search	
Last checkpoint time:	Tue Dec 03 02:13:07 GMT 2019	
Current status:	Not running	
Errors detected:	No errors	
Fingerprint collision probability:	calculated: .041, observed: .0088	

Figure 1. Expected successful model checking of Hermes.

## A.13 Notes

**Running-out of disk space and constants.** A long running TLC process writes unexplored states to the disk for later exploration. Thus, a model with very large constants, can use up all of the available disk space, resulting in a TLC crash. If this occurs, please try to acquire more disk space or uses smaller model constants.

**Model check Hermes protocol with RMWs.** In step 4 of §A.11 simply replace "HSpec" to "HRSpec".