
ccnSim-0.4-Parallel

Installation and User Guide

Michele Tortelli and Dario Rossi

November 21, 2017

This introductory guide provides the key steps for the installation of the newly released `ccnSim-Parallel` [2], along with a couple of sample use cases. `ccnSim-Parallel` proposes to, insted of slicing nodes of the network over multiple cores, to slice independent portions of the catalog over multiple cores. In contrast to network slicing, which would incur significant MPI overhead, the new technique exhibit an ideal speedup in the number of cores, enabling the simulation of web-scale scenarios (i.e., with catalogs comprising 10^{12} contents) with exiguous amount of memory and CPU times. The new parallel simulation engine that achieves $100\times$ gain over the `ModelGraft` engine[6] (available in `ccnSim-0.4`) and thus a $10000\times$ gain over the event-driven simulation (default in `ccnSim-0.4`)! For a thorough description of the single-threaded `ModelGraft` technique please refer to [6], while for more details on the background at the bases of `ccnSim-Parallel` we point the reader to our technical report [7].

Contents

1	Background	2
1.1	Theory	2
1.2	Practice	2
2	Installation	3
2.1	Portability	3
2.2	Akaroa2	4
2.3	Omnet++	4
2.4	<code>ccnSim-Parallel</code>	5
3	Example-1: Trillion-scale simulation made possible!	5
3.1	Scenario, bare-metal, and scripts	6
3.2	Results	7
4	Example-2: <code>ccnSim-Parallel</code> vs <code>ModelGraft</code> vs Classic ED Simulation	8
4.1	Results	9
5	Customization	9
5.1	General Information	9
5.2	Applicability	9
5.3	Encoding of Messages Exchanged between Master and Slaves	9

1 Background

1.1 Theory

From a *theoretical viewpoint*, ccnSim-Parallel is designed around an innovative technique for parallelizing the simulation of general cache networks, referred as *catalog slicing* (CS-POST-MT). One of the classic ways for parallelizing simulations is that of geographically splitting the network into non-overlapping regions, and then mapping their simulation to multiple cores (or threads); it is a well-investigated approach that generates a significant communication overhead due to “message passing”, especially in cache networks where requests’ propagation creates strong correlation between the state of neighboring cache nodes. With the *catalog slicing* technique, instead, each core *independently* simulates the whole topology and the whole downscaled catalog (with an alternative and more effective technique with respect to our previous ModelGraft *downscaling* technique [6]), but only for a fraction of time, before sending aggregated information to the master node.

In particular, the instantiation of multiple and independent parallel threads is made possible thanks to Che’s approximation [3], according to which in a network of Least Recently Used (LRU) caches, dynamics of different contents can be considered as both spatially and temporally independent; as a consequence, distinct threads can be assigned to different portions of the request process timeline, thus sizing the global simulation time for each thread according to the maximum number of available threads. An elected *master* thread, then, will simply collect and sum statistics obtained by the different *slaves* in order to compute the global network state and decide on both *dynamic transient state* and *simulation end*. All desirable properties of ModelGraft (e.g., convergence to a consistent state) also hold for ccnSim-Parallel.

This approach:

- (i) avoids heavy communication overhead related to message passing, leading to a *nearly proportional* gain with the number of parallel threads (up to ~32 threads), and
- (ii) it is slightly more accurate with respect to the single-threaded ModelGraft engine [6] (0.1% error as opposite to 1% error)
- (iii) introduces an additional speedup of about *2 orders of magnitude* with respect to the single-threaded ModelGraft engine [6]
- (iv) achieves an overall speedup of about *4 orders of magnitude* with respect to the optimized event driven engine [4]

For a thorough description and evaluation of the new ccnSim-Parallel technique, referred to as CS-POST-MT, please refer to our technical report [7].

Note: Given that “ModelGraft” is a cool name, whereas “CS-POST-MT” is not, in the following, by abuse of language we may be tempted to colloquially refer to CS-POST-MT as the “Parallel ModelGraft”. There is however a subtle abuse here, since the original ModelGraft[6] implements a CS-PRE-ST (where ST stands for Single-Threaded) technique according to terminology in [7], that can be parallelized as CS-PRE-MT (where MT stands for Multi-Threaded); however, since CS-PRE-MT technique is less efficient than CS-POST-MT (where PRE vs POST refer to specific *a priori* vs *a posteriori* binning techniques), the ccnSim-0.4-Parallel ultimately offer only the most efficient CS-POST-MT implementation, that we refer to “Parallel ModelGraft CS-POST-MT” to avoid ambiguities or also simply “Parallel ModelGraft”. Sorry if this leads to ambiguities!

1.2 Practice

The implementation of ccnSim-Parallel involves the integration of two different softwares:

- (i) the open source ccnSim-v0.4 simulator [2], which includes an implementation of the single-threaded ModelGraft technique (CS-PRE-ST in [7]) as well as the newly proposed multi-threaded CS-POST-MT technique, and

- (ii) the licensed Akaroa2software [5, 1], which provides APIs for the design of a master-slave infrastructure.

ccnSim-Parallel requires to make modification to both pieces of software in order to ease their integration. For the sake of usability, we provide here a unique and dedicated version: to be noted that due to the strong integration between the two aforementioned softwares, ccnSim-Parallel requires the modified Akaroa2. It is important to notice that users *should obtain a specific license* for the download of the original Akaroa-2.7.13, i.e., the initial version used for the deployment of ccnSim-Parallel: a specific *patch* (`Akaroa_2.7.13_for_PMG.patch`), is instead distributed in the GitHub along with the ccnSim-Parallel code, in order to apply the required modifications.

2 Installation

This section covers all the steps needed for the installation of ccnSim-Parallel. We suggest the reader to check and execute them by following the presentation order.

2.1 Portability

ccnSim-Parallel has been tested on the following platforms, and with the following softwares:

- Ubuntu Linux 14.04 (64-bit)
- Ubuntu Linux 16.04 Server (64-bit)
- Omnetpp-4.6
- Omnetpp-5.0
- Akaroa-2.7.13

Prerequisites

- **Boost libraries ≥ 1.54 :** they can be installed either by using the standard packet manager of your system (e.g., apt-get install, yum install, port install, etc.), or by downloading them from <http://www.boost.org/users/download/>, and following instructions therein.
- **gcc $> 4.8.1$:** on Ubuntu platforms, gcc can be updated by adding the `ubuntu-toolchain-r/test` PPA. Sample commands for the latest 5.x version are:

```
sudo add-apt-repository ppa:ubuntu-toolchain-r/test
```

```
sudo apt update
```

```
sudo apt install gcc-5 g++-5
```

```
sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-5 60
```

```
--slave /usr/bin/g++ g++ /usr/bin/g++-5
```

- **parallel-ssh:** in order to ease the simulation workflow, a passwordless SSH login needs to be present on all the available servers. *parallel-ssh* can than be used in order to send commands to slaves from the master node.

```
sudo apt-get install pssh
```

2.2 Akaroa2

As mentioned above, Akaroa2 requires a specific license in order to be used. For instructions about software download and licenses, please visit the website <https://akaroa.canterbury.ac.nz/akaroa/>. The design of ccnSim-Parallel, concerning the Akaroa component, started from the Akaroa-2.7.13 version. The whole set of modifications that have been made in order to implement the parallel ModelGraft strategy are grouped inside the *Akaroa_2.7.13_for_PMG.patch* patch (available in the same GitHub project of the present manual https://github.com/TeamRossi/ccnSim-0.4-Parallel/blob/master/Akaroa_2.7.13_for_PMG.patch).

Once obtained the original licensed Akaroa-2.7.13, place it in the same directory with the aforementioned patch, and type:

```
patch -s -p0 < Akaroa_2.7.13_for_PMG.patch
```

Notice that the original Akaroa directory keeps the same name even after the application of the patch.

Once the patch is applied, type the following commands from the root directory of the patched Akaroa in order to install it:

```
./configure  
make  
sudo make install
```

Akaroa binaries, libraries, and include files will be installed in `/usr/local/akaroa/bin`, `/usr/local/akaroa/lib`, and `/usr/local/akaroa/include`, respectively. If you *do not have sudo privileges*, and you need to install Akaroa into another location, e.g., `<install_path>`, please add this location to your `PATH` variable, and use the following commands:

```
./configure --prefix=<install_path>  
make  
make install
```

Furthermore, always in the case of lacking `sudo` privileges, users might need to modify `LIB` and `PYTHON` paths in the several “`Makefile.config_*`” files and inside the “`pyconfig.py`” file according to the customized locations of related libraries, before compiling and installing Akaroa.

2.3 Omnet++

ccnSim-Parallel has been tested with both *Omnetpp-4.6* and *Omnetpp-5.0*, which are available at <https://omnetpp.org/>. Before the installation, Omnetpp-x requires prerequisite packages to be installed. For the complete list, please read the official installation manual. Furthermore, since Omnetpp-x `bin/` directory needs to be added to the `PATH` variable, the following line should be put at the end of the `~/.bashrc` file:

```
export PATH=$PATH:$HOME/omnetpp-x/bin
```

After having closed and saved the file, please close and re-open the terminal. Regardless of the Omnetpp version, a couple of patch files need to be applied to it before compilation and installation. These patch files are provided with ccnSim-Parallel, which we suppose being correctly downloaded (see next section) and decompressed in the `$CCNSIM_DIR`. Provided that Omnetpp is present in the `$OMNET_DIR`, specific instructions to patch, compile, and install it without the graphical interface (not needed for parallel ModelGraft simulations) are provided in the following, according to the selected version:

- Omnetpp-4.6

```
pint:~$ cd $CCNSIM_DIR
pint:CCNSIM_DIR$ cp ./patch/omnet4x/ctopology.h $OMNET_DIR/include/
pint:CCNSIM_DIR$ cp ./patch/omnet4x/ctopology.cc $OMNET_DIR/src/sim/
pint:CCNSIM_DIR$ cd $OMNET_DIR/
pint:OMNET_DIR$ NO_TCL=1 ./configure
pint:OMNET_DIR$ make
```

- Omnetpp-5.0

```
pint:~$ cd $CCNSIM_DIR
pint:CCNSIM_DIR$ cp ./patch/omnet5x/ctopology.h $OMNET_DIR/include/omnetpp
pint:CCNSIM_DIR$ cp ./patch/omnet5x/ctopology.cc $OMNET_DIR/src/sim
pint:CCNSIM_DIR$ cd $OMNET_DIR/
pint:OMNET_DIR$ ./configure WITH_TKENV=no
pint:OMNET_DIR$ make
```

Compilation flags can also be set through the file *configure.user* (e.g., to disable Qtenv).

2.4 ccnSim-Parallel

As anticipated above, the .tgz file containing the ccnSim-Parallel code can be obtained from <http://perso.telecom-paristech.fr/~drossi/ccnSim>. After having decompressed it in the \$CCNSIM_DIR folder, installation requires the following steps:

```
./scripts/makemake.sh
make
```

If all the steps described in the previous sections have been successfully executed, simulations of general cache networks using the parallel ModelGraft technique can now be launched. The several steps needed to simulate sample scenarios will be described in the following.

3 Example-1: Trillion-scale simulation made possible!

This section describes all the steps needed to simulate a Web-scale scenario, i.e., a catalog comprising a trillion contents. It is an extreme scenario which highlights the capabilities of the new parallel CS-POST-MT technique. For an extended and thorough performance evaluation, which considers scenarios with different cardinalities, we refer the reader to our technical report [7]. This section guides you to the basic in order to be able to easily run the scenarios shown in [7]. Notice that while CS-POST-MT differ from ModelGraft (CS-PRE-ST), for abuse of language we indicate as “Parallel ModelGraft” a CS-POST-MT strategy (and not a CS-PRE-MT which would be inefficient as shown in [7]).

3.1 Scenario, bare-metal, and scripts

Scenario. We consider a CDN-like topology composed of an Abilene core network, and several 4-level binary trees attached to it, thus representing access networks. By using the resulted topology, which comprises 67 nodes in total, we simulate an initial non-downscaled catalog of $M = 10^{12}$ contents, and through an Independent Reference Model (IRM), i.e., i.i.d. requests, we consider $R = 10^{12}$ total requests as a non-downscaled initial point. We consider nodes having non-downscaled Least Recently Used (LRU) caches of $C = 10^8$ contents. By following guidelines provided in [6], we set the *downscaling* factor of the ModelGraft technique at $\Delta = 10^7$ (for which the downscaled cache size of the equivalent TTL scenario is expected to be $C' = C/\Delta = 10$, that is both robust and efficient). As a consequence, we effectively simulate a *downscaled* catalog of $M' = M/\Delta = 10^5$ contents, with Time-to-Live (TTL) caches with a downscaled cache size of $C' = 10$, and we generate a downscaled number of $R' = 10^7$ requests.

Bare-metal. The dedicated cluster used for the experiment comprises $NS = 3$ Cisco UCS-B series servers, each one hosting 2 NUMA nodes, with a Xeon E5-2690 CPU, $NC = 12$ physical cores per NUMA node (i.e., 48 CPUs in total with hyperthreading) operating at 2.60GHz, and 378 GB of RAM memory. For ease of description we call our servers “modelgraftX”, where $X \in \{1, 2, 3\}$, and we suppose that a *modelgraft* user is present in each of them. We also remind that it is important to have a passwordless SSH login on all the available servers. For this particular scenario, we report results related to simulations done with $NT = 64$ parallel threads and $NS = 2$ physical servers.

Scripts. The main file used to launch the simulation of the described scenario is:

```
① ./launch_Parallel_ModelGraft_1e12.sh
```

This *bash script* will appear inside the Akaroa folder after having applied the provided Akaroa_2.7.13_for_PMG.patch patch. In order to make it executable you need to type `chmod +x launch_Parallel_ModelGraft_1e12.sh`. It is important to notice that:

- The script is supposed to be executed from one of the servers used for the simulation, meaning that both Akaroa and ccnSim-Parallel are installed on the same machine. If executed from a different machine than the servers used for the simulations, the script will need some modifications.
- The script has been conceived considering the specified pool of 3 servers, i.e., modelgraft1, modelgraft2, and modelgraft3, with a common “modelgraft” user present on all of them;
- If using different servers and username, the script needs to be modified accordingly. Sometimes it is more practical to use server names instead of IP addresses; in order to do that, entries like “x.x.x.x servername” should be added in the “/etc/hosts” file of all the servers.

We strongly suggest to carefully read both the extensively documented script ① for a detailed description of all its components and commands, and the User Manual of ccnSim-v0.4 [2] for insights related to its structure and to the ModelGraft technique.

In summary, script ① will:

- I. specify the number of *physical servers* and *parallel threads* that will be used to run the parallel ModelGraft;
- II. set all the parameters needed to define the simulated scenario;
- III. automatically probe and allocate resources on the available servers according to their load (i.e., the least loaded ones are considered);

IV. launch Akaroa daemons (both Master and Slaves) on the reserved resources;

```
${akaroaBinDir}/akmaster &
```

```
${akaroaBinDir}/akslave &
```

V. call the script “runsim_script_Parallel_ModelGraft.sh” dedicated to ccnSim-Parallel simulations.

```
(2) ./runsim_script_Parallel_ModelGraft.sh {parameters}
```

All the scenario parameters defined within script (1) are passed in the form of command line parameters to script (2), which is located in `$CCNSIM_DIR`. For a complete list of all the possible parameters please refer to the User Manual of ccnSim-v0.4 [2].

We strongly suggest to carefully read the extensively documented script (2) for a detailed description of all its components and commands, and the User Manual of ccnSim-v0.4 [2] for insights related to its structure and to the ModelGraft technique.

In summary, script (2) will:

- I. check if the required T_C file is already available; otherwise it will create a new one with *random* T_C values, which will be distributed to all the available servers;
- II. create a new .ini file according to {parameters};
- III. launch the parallel ModelGraft simulations through the Akaroa APIs;
- IV. collect and elaborate results.

Point I. is worth to be discussed: since the ModelGraft technique [6] makes use of TTL caches with an eviction timer set according to the *characteristic time* T_C [3] of the respective LRU caches, files reporting the T_C value of each node in the simulated topology are needed as input. Since in most of the real scenario users do not know T_C values a priori, the ModelGraft technique is able to iteratively converge to a consistent state through a feedback loop, even when accepting random T_C values as input. As a consequence, script (2) will firstly check if a user-provided T_C file is already present, and if not, it will randomly generate values for all the nodes, collect them in one file, and distribute it to all the available servers (since all the threads instantiated over multiple servers need to have a common T_C file as a starting point). This file will be automatically erased from all the servers if randomly generated (i.e., user-defined ones will not be erased).

As for the management of the output files produced by all the parallel threads, the policy adopted by the current version of script (2) is that of *removing them all from each server, thus saving only a SUMMARY file*, namely “ALL-MEASURES*”, which will be kept in the `${CCNSIM_DIR}/${resultDir}/parallel` folder of the server where script (2) has been executed from (which, by default, is modelgraft1). If you want/need to change this policy, please follow instructions provided in the script (2) file.

3.2 Results

Results are presented in Fig. 1, which is a stripped down version of the more large scale and extensive tests carried out in [7] (see Fig.10 in [7]). The figure reports both CPU time and memory occupancy on a logarithmic axis; in particular, results related to three different strategies are compared: classic event-driven (ED) simulation, single-thread ModelGraft (CS-PRE-ST), and the last parallel ModelGraft (CS-POST-MT) technique (see [7] for exact definition of CS-PRE-ST and CS-POST-MT). As it can be noticed, simulating such a huge scenario would be prohibitive if

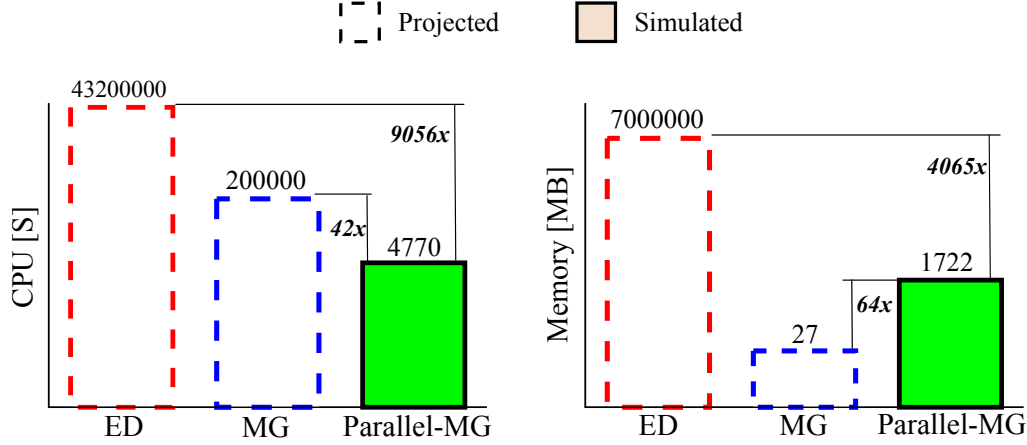


Figure 1: Parallel ModelGraft vs ED and ModelGraft projections: CPU time and Memory occupancy for Web-scale scenario ($M = 10^{12}$, $R = 10^{12}$, $C = 10^8$, $\Delta = 10^7$, $NT = 64$ parallel threads, $NS = 2$ physical servers for the parallel ModelGraft technique).

relying on the classic event-driven approach, considering that we would need more than one year of CPU time and more the 7 TBytes of RAM. That is the reason why we report only projected results, computed from linearly interpolating results obtained for smaller scenarios. Since the introduction of the single-threaded ModelGraft technique [6], instead, memory occupancy is not considered as a bottleneck anymore (notice that only 27 MB are required); as a consequence, CPU time represented the only bottleneck (i.e., for this scenario, we would need more than 2 days of simulation).

The new parallel CS-POST-MT technique, indeed, aims at overcoming the CPU bottleneck by parallelizing the simulation over multiple threads and physical servers. In this particular case, by instantiating $NT = 64$ parallel threads over $NS = 2$ physical servers, we notice from Fig. 1 a dramatic decrease in the CPU time, i.e., $9056\times$ and $42\times$ compared to classic ED simulation and single-threaded ModelGraft, respectively. As for the memory occupancy, instead, we can observe that while parallelizing the simulation by creating exact replicas of network and catalog increases the required memory, the total amount is still considerably slow and it can also be split over multiple servers. At the same time, the parallel ModelGraft CS-POST-MT technique still considerably reduces the memory occupancy w.r.t. ED simulation by up to $4065\times$.

4 Example-2: ccnSim-Parallel vs ModelGraft vs Classic ED Simulation

The Web-scale scenario seen before is treatable only by using the scalability and performance of the parallel ModelGraft CS-POST-MT technique, meaning that we actually miss other points of comparison. As a consequence, in this section we compare the new parallel ModelGraft technique against the previous single-threaded one, and against the classic event-driven (ED) simulation (both available with `ccnSim-v0.4` [2]), by considering a smaller scenario as a reference.

In particular we simulate a 4-level binary tree (i.e., 15 nodes) topology, with a non-downscaled catalog cardinality of $M = 10^9$, cache size $C = 10^6$, $R = 10^9$ total requests, and a downscaling factor of $\Delta = 10^5$. This scenario can be simulated by using the provided script

```
./launch_Parallel_ModelGraft_1e9.sh
```


4.1 Results

Results are reported in Tab. 1, which is a stripped down version of the more large scale and extensive tests carried out in [7] (see Tab.II at the end of [7]). Results allow to gather three main considerations:

- (i) not only the parallelization of the ModelGraft CS-POST-MT technique does not introduce further inaccuracies, but is furthermore *more accurate* than ModelGraft CS-PRE-ST technique of [6],
- (ii) allows to *drastically reduce the CPU time* of a further order of magnitude with respect to the single-threaded ModelGraft, (notice that this small-scale scenario allows to directly compare with ED but provides conservative gains for CPU reduction, since $\Delta = 10^5$ is small w.r.t larger scenarios)
- (iii) still *requiring a reasonable amount of memory* with respect to the number of instantiated parallel threads, i.e., 32 in this case (it is worth considering that the total memory demand can be spread over multiple machines when multiple servers are available).

5 Customization

5.1 General Information

Users interested in experimenting and ameliorating the implementation of the parallel ModelGraft CS-POST-MT technique can refer to the CHANGELOG.txt file (present in both patched Akaroa folder and ccnSim-Parallel folder) where a list enumerating all the modified files is reported. The goal is that of facilitating the orientation of newcomers into both Akaroa and ccnSim-Parallel codebases. Nevertheless, for a deeper understanding of all the elements included in the parallel ModelGraft technique, we strongly encourage users to read (i) Akaroa documentation [1], ccnSim-v0.4 documentation [2], ModelGraft paper [6], the parallel technique technical report [7], and all the well documented scripts present in both patched Akaroa folder and ccnSim-Parallel folder.

5.2 Applicability

For a complete list and discussion on all the possible scenarios the parallel ModelGraft technique can be used for, please refer to Section 3.1 of the ccnSim-v0.4 manual [2].

5.3 Encoding of Messages Exchanged between Master and Slaves

The patched Akaroa-2.7.13 needs to define a MAX_MSG_LENGTH variable for the messages which are exchanged between Master and Slaves. In addition, one of the modifications introduced in order to let Akaroa and ccnSim-Parallel communicate and be compliant with each other, is that of including a *vector*, namely “HitMissVector”, inside the exchanged messages. This vector should contain info about hit and miss events collected by the parallel and independent slaves, which are then sent to the Master node.

For the ease of implementation, both MAX_MSG_LENGTH and HitMissVect have been sized according to the biggest simulated topology. In practice, since each node in the network will send $2 \times engineWindow = 2 \times (W/NT)$ Hit and Miss samples, where W is the initially dimensioned window size, and NT is the number of allocated parallel threads, the size of the *aggregated* HitMissVect sent by each parallel thread will be equal to $vectSize = N \times 2 \times engineWindow$, where N is the total number of nodes. Since $W = 100$ by default, and since the largest topology that we simulated is the CDN-like one (i.e., with 67 nodes), the aforementioned variables are set at compile time as:

```
char HitMissVect[57000] (inside 'src/include/checkpoint.H' file of Akaroa)
```

Table 1: Parallel ModelGraft vs ModelGraft vs Event-driven simulation: accuracy loss, CPU and memory gain for a very large scenario ($M = 10^9$, $R = 10^9$, $C = 10^6$, $\Delta = 10^5$, $NT = 32$ parallel threads, $NS = 1$ physical server for the parallel ModelGraft technique)

Cache Decision Policy	Technique	p_{hit}	Loss	CPU time	Gain	Mem [MB]	Gain
LCE	ModelGraft	33.1%	0.1%	143 s	286x	23	277x
	Event-driven	33.2%		11.4 h		6371	
	Parallel ModelGraft	33.4%	0.2%	13.8 s	2974x	829	7.7x

#define MAX_MSG_LEN 57500 (inside src/ipc/connection.H' file of Akaroa)

The HitMissVect size (in Bytes) has been obtained by $N \times 2 \times engineWindow \times 4$, supposing $NT = 1$ (i.e., the case with the biggest vector). The exact value should be 53600 Bytes, so the used one is slightly increased for safety reasons. The size of the MSGs that should carry also the HitMissVect is, then, set accordingly to $MAX_MSG_LEN = 57500$ (i.e., 500 Bytes more to reserve space for other values transmitted within the same message).

It is IMPORTANT to notice that, if using other values then $N = 67$, $W = 100$, which bring to a bigger HitMissVect than 57000 Bytes (the equation is always $N \times 2 \times (W/NT)$), please modify the relative *checkpoint.H* and *connection.H* files, and recompile Akaroa (make; sudo make install). If smaller values are set w.r.t. the real ones, segmentation fault problems might happen.

References

- [1] Akaroa Project Website. <https://akaroa.canterbury.ac.nz/akaroa/>.
- [2] ccnsim website. <http://perso.telecom-paristech.fr/~drossi/ccnSim>.
- [3] H. Che, Y. Tung, and Z. Wang. Hierarchical web caching systems: Modeling, design and experimental results. *IEEE JSAC*, 20(7):1305–1314, 2002.
- [4] R. Chiocchetti, D. Rossi, and G. Rossini. ccnSim: an highly scalable ccn simulator. In *IEEE ICC*, 2013.
- [5] G. Ewing, K. Pawlikowski, and D. McNickle. Akaroa-2: Exploiting network computing by distributing stochastic simulation. *SCSI Press*, 1999.
- [6] M. Tortelli, D. Rossi, and E. Leonardi. A hybrid methodology for the performance evaluation of internet-scale cache networks. *Elsevier Computer Networks, Special Issue on Softwarization and Caching in NGN*, 2017.
- [7] Michele Tortelli, Dario Rossi, and Emilio Leonardi. Parallel simulation of very large-scale general cache networks. <http://www.enst.fr/~drossi/paper/rossi17parallel-cache.pdf>, November 2017. keyword=newnet,ccn.