Multi-threaded Simulation of 4G Cellular Systems within the LTE-Sim Framework

Alessandro Pellegrini
Dipartimento di Ingegneria Informatica
Automatica e Gestionale
Sapienza, University of Rome
pellegrini@dis.uniroma1.it

Giuseppe Piro
Dipartimento di Elettrotecnica ed Elettronica
Politecnico di Bari
Bari, Italy
g.piro@poliba.it

***Abstract*—Nowadays, an always increasing number of researchers and industries are putting a large effort in the design and the implementation of protocols, algorithms, and network architectures targeted at the the emerging 4G cellular technology. In this context, multi-core/multi-processor simulation tools can accelerate their activities by drastically reducing the time required to simulate complex scenarios. Unfortunately, today's available tools are mostly single-threaded and they cannot exploit the performance gain offered by parallel programming approaches. To bridge this gap, we have significantly upgraded the LTE-Sim framework by implementing a concurrent scheduling algorithm, namely the Multi-Master Scheduler, aimed at efficiently handling events in a parallel manner, while guaranteeing the correct execution of the simulation itself. Experimental results will demonstrate the effectiveness of our proposal and the performance gain that can be achieved with respect to other classical event scheduling algorithms.**

I. Introduction

The widespread use of new generation mobile devices, together with the increasing popularity of Web 2.0 and cloud applications, will lead to a massive gain of data traffic from/to mobile users [1]. In order to satisfy these requirements, and guarantee a good level of Quality of Service (QoS) offered to mobile users, the 3rd Generation Partnership Project (3GPP) has introduced Long Term Evolution (LTE) and LTE-Advanced (LTE-A) specifications as the next step of the current 3G mobile networks [2].

The optimization of all LTE and LTE-A aspects is a topic worth of investigation for both the industry and academic communities. Moreover, current research trends in the LTE field entail simulations of complex scenarios, involving several cells and hundreds (if not thousands) of users. At the present time, the only valuable tools are those developed for NS-3 within the LENA project [4] and LTE-Sim [5]. Unfortunately, due to the high detail of models they provide, these simulators require a huge computational power. In addition, since they have been conceived as a single-thread process, the time needed for carrying out complex studies mat be unacceptably large. In fact, it would be desirable to develop more efficient simulation platforms able to exploit the high computational performance offered by emerging multi-core computers.

To bridge this gap, we have significantly upgraded the LTE-Sim framework by implementing a concurrent scheduling algorithm, namely the *Multi-Master Scheduler*, which is able to efficiently schedule those events which can be concurrently executed, thus allowing a parallel execution of the simulation, while ensuring its correctness, at the same time. Experimental results will show the good impact on scalability of the implementation within the LTE-Sim platform, providing a relevant benefit on research activities on the LTE-related fields.

The rest of the paper is organized as in the following: Sec. II describes the LTE-Sim open source framework, highlighting in what aspects it can be enhanced for supporting a multi-threaded execution. Sec. III presents our conceived Multi-Master scheme. Sec. IV presents some significant results for demonstrating the behavior of the implemented solutions in some reference scenarios. Finally, Sec. V draws the conclusion.

II. Simulating LTE Cellular Systems with LTE-Sim

A. Main features covered by LTE-Sim

LTE-Sim is an emerging open source tool conceived for simulating LTE and LTE-A networks. Its main features have been summarized in [5]–[7]. It supports single and heterogeneous multi-cell environments, QoS management, multi-users environment, user mobility, handover procedures, frequency-reuse techniques and several other aspects related to the LTE technology.

In LTE-Sim, the network topology is composed by a set of cells and a number of the following network nodes: User Equipment (UE), evolved Node B (eNB), Home eNB (HeNB), and Mobility Management Entity/Gateway (MME/GW) [5], [7]. Each of them is identified by a unique ID, whereas its position is defined in a cartesian system. Several mobility models, such as constant position, random way point, random direction, and manhattan mobility model, are supported for allowing mobile terminals to move into the network.

At the application layer, four different traffic generators have been developed: video, VoIP, CBR, and infinite buffer. Moreover, several functionalities of both user-plane and control-plane protocol stacks are present.

Channel and PHY models have been developed according to 3GPP specifications. In particular, the LTE radio access is based on Orthogonal Frequency Division Multiplexing (OFDM) and provides a highly flexible bandwidth (from

1.4 to 20 MHz). Both frequency-division duplex (FDD) and time-division duplex (TDD) multiple-access techniques are supported. At the PHY layer, radio resources are allocated among users in a time-frequency domain [2]. In the time domain, radio resources are distributed at every Transmission Time Interval (TTI), each one composed by two consecutive time slots of 0.5 ms. In the frequency domain, instead, the whole bandwidth is divided into 180 KHz sub-channels. A time/frequency radio resource spanning over one 0.5 ms time slot in the time domain and over one sub-channel in the frequency domain is called Resource Block (RB) and corresponds to the smallest radio resource that can be assigned to a UE for data transmission.

At the beginning of each TTI, each base station is in charge of performing the radio resource allocation procedure among the users it serves. In order to offer a valid support to researchers working in the radio resource management research field, LTE-Sim implements several well-known scheduling strategies for both the downlink and the uplink [8]. They distribute radio resources among mobile users by taking into account, among other parameters, also the channel quality experienced by them in the downlink and uplink link.

Several propagation-loss models, which consider path-loss impact, penetration loss, shadowing, and fast fading in a wide range of urban, sub-urban, and rural environments, as well as an accurate physical error model based on the estimation of the Block Error Rate (BLER), have been also implemented.

B. Event management in LTE-Sim

The main goal when depicting the characteristics of a multi-threaded simulation engine is the throughput maximization, i.e. enhancing the number of events which can be concurrently executed. Starting from this assumption, it is very important to note that the design of a multi-threaded extension of the presented LTE-Sim platform should be carried out considering the generation, the management, and the interaction of events during the simulation.

In LTE-Sim, any operation executed within a network (e.g. generation of a packet at the application layer, physical transmission, radio-resources allocation, handover procedure, ...) is modeled, in the vision of an event-driven architecture of the simulator, as an event. Hence, each event is in charge of performing a specific activity during the simulation and it is (impulsively) executed at a specific time instant.

In order to identify the most efficient way to concurrently execute events, we have to analyze (i) the number of events that have the same timestamp during the simulation, (ii) the relation between events marked with the same timestamp, and (iii) the interaction between events that can be executed progressively in the time.

To this end, focusing the attention on a simple LTE scenario composed by one macro cell and a number of users in the range [50, 150], we report in Fig. 1 both the Cumulative Distribution Function (CDF) of the number of generated events with the same timestamp, and their execution times.

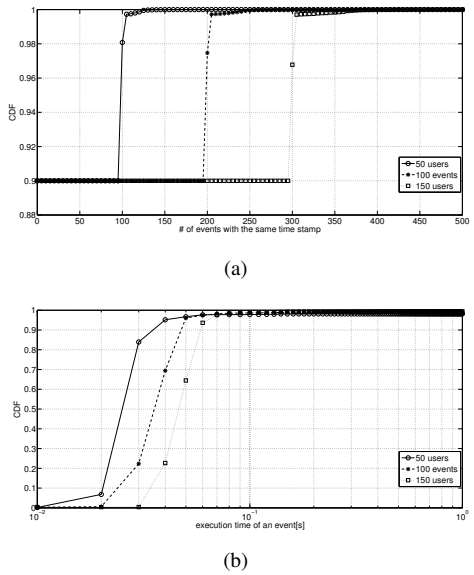

Fig. 1. CDF of (a) the number of events with the same time stamp and (b) their execution time in a scenario with a variable number of users.

It is clear that—with a very high probability—a number of events slightly greater than the double of the users' quota shows the same timestamp. This result is due to radio-resources' distribution in a LTE network: Most of the events are executed at the beginning and at the end of each TTI (the reader is encouraged to think to the scheduling strategy, the packet transmission, and the packet reception). Within the TTI, only a few number of events can be generated, such as the generation of the packet at the application layer, the bearer initialization, etc.

In order to ensure the correct execution of the simulation, we should note that all events with the same timestamp are safe to be concurrently executed, since they are independent from each other (i.e. their execution involves updating different data structures within the simulation model's state). On the other hand, events with different timestamps must be executed in timestamp order because the execution of one event could influence the system behavior in the future, and out-of-order execution is likely to lead to wrong results.

We found that the average time required for executing an event grows with the number of users into the network. The reason is that a higher number of mobile terminals requires a higher computational load for those events performing the radio-resource scheduling algorithm.

All of these findings will be exploited for selecting the most suitable multi-threading architecture for the considered network simulator.

III. Multi-threaded Scheduling and Execution

When dealing with parallel and concurrent algorithms, three key factors must be explicitly addressed, in order to efficiently exploit the increased computational power provided by an enlarged number of available processing units: i) *software*

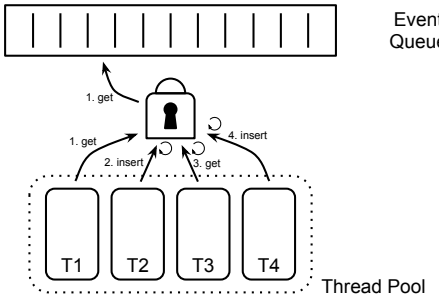

Fig. 2. Symmetric Synchronization

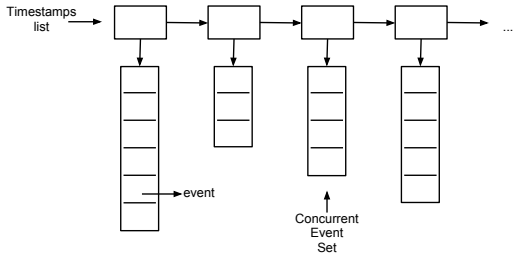

Fig. 3. Concurrent Event Queue

contention, i.e. the amount of shared data which can be concurrently accessed by different threads, therefore requiring some sort of serialization (e.g. the definition of critical sections relying on locking primitives); ii) *hardware contention*, i.e. the effects on the memory architecture (entailing, e.g., bus accesses, cache coherency protocols) when performing concurrent operations on shared data; iii) the *degree of parallelism* provided by the application being parallelized. This is extremely true in the context of simulation, where a reduced set of data structures is used to maintain the event queue during the execution (if compared, e.g., with the amount of data structures used by operating systems' kernels).

If from one side the degree of parallelism exhibited by an application can be very difficult to be captured, on the other hand efficiently tuning the parallel application in order to reduce contention at the minimum can avoid thrashing even when the number of computational resources being used is higher than actually needed.

Symmetric synchronization is a classical approach towards multi-threaded simulation. As shown in Fig. 2, a thread pool composed by n threads must access the sole event queue (which maintains timestamp-ordered events) due to two main typologies of actions, namely *insert* and *get*. In order to ensure event-queue consistency, accesses to this shared resource must be serialized through the usage of some locking primitive. This logical contention wastes computational resources and produces secondary effects (i.e. on memory bus, if synchronization is enforced by relying on spinlocks) to an extent unbearable in the context of high performance simulations.

?! As mentioned in Sec. II, the time required for simulating an event is always lower than 0.1s,therefore an efficient implementation of a concurrent scheduling algorithm must guarantee that the cost for selecting the next event(s) to be executed is likely less than an event's execution, in order to provide any benefit. For this reason, we have explicitly discarder the adoption of classical symmetric strategies. Instead, we propose a novel/specifically-targeted strategy, named the Multi-Master Scheduler.

By *subsystem* we define the point of the protocol stack in which an event can be generated within the LTE-Sim platform. Considering the number of entities forming up the LTE protocol stack and the presence of several nodes into the network, it is easy to understand that events can be generated inside a wide range of subsystems. As reported in Sec. II, most of these events are executed at the beginning/end of the TTI. Moreover, events marked with the same timestamp can be safely executed in parallel. Hence, we can introduce the following properties:

Property 1. *If two events e and e' belong to two different subsystems, then they access separate portions of the simulation state.*

Property 2. *During the execution of a certain event e associated with a simulation time T_e, we have that $\forall e'$ generated during the execution of e, $T_{e'} > T_e$.*

We therefore rely on the notion of *worker thread*, which has been recently shown [**?**], [9] to be a viable means for supporting the parallel execution of simulation models[1]. In particular, to efficiently exploit the parallel computational power, the scheduling algorithm which we hereby propose will be able to:

A) return in constant time the whole set of concurrent events (i.e. all the events associated with the same timestamp);
B) allow wichever worker thread to be recognized by any thread as the master thread, thus taking care of events assignment to other worker threads, even if they are still executing events associated with a previous timestamp.

To fully enforce point A), we propose to organize pending events (i.e. events which have been already generated, but still belong to a timestamp in the future) into a *Concurrent Event Queue* as depicted in Fig. 3. In particular, we propose to group all the events marked with the same timestamp T_s into the same *Event Set*, and link the various Event Sets into a *Timestamp List* ordered according to an increasing timestamp value. Given this definition, we can introduce the following operations to manipulate the aforementioned data structure:
Event Insertion: Upon the generation of a new event e' associated with timestamp $T_{e'}$ during the execution of an event e, the Timestamp List is scanned to determine into which Event Set (if any) the new event must be placed, i.e. the Event Set S such that $T_s = T_{e'}$. If no suitable set is found (i.e. there is no set associated with the timestamp $T_{e'}$), a new set is created and linked in a position such that the timestamp ordering in the list is maintained. On the other hand, the event

[1]Nevertheless, the proposals in [**?**], [9] target an optimistic simulation framework [10], while we explicitly deal with conservative simulations.

e is simply inserted into the related Event Set. This operation has an $O(n)$ cost, n being the number of Event Sets in the list, in the worst case.

Event Scheduling: When the scheduling algorithm is run for selecting events to be executed, they are extracted from the first set in the list, and two cases might arise:

1) the number of concurrently-running threads is larger than concurrent events available in the first set;
2) the number of concurrently-running threads is smaller than concurrent events available in the first set.

If case 1) is enforced, then the current master thread, which is executing the scheduling operations, assigns a subset of events to each thread, which will execute them in parallel when the tasks currently being carried out (if any) are completed. On the other hand, case 2) entails a traditional execution by the master thread, where each event is retrieved from the Event Queue (i.e. from the first Event Set), executed and then removed. This choice is related to the fact that, during the scheduling phase, the master thread is the only worker thread which we are sure is not currently in charge of executing any simulation event. In fact, determining which are the idle threads among the ones in the pool is a costly operation requiring some synchronization effort. At the same time, a thread currently becomes the master only if it has finished its work assignment. Therefore, since point 2) is enforced if the number of available events for a given time set is limited, given the medium granularity of events' durations, it is a more convenient trade-off to immediately execute events, rather than deciding which worker thread will be in charge of executing them in the near future.

In either case, since events are executed in incresing timestamp order, either if one event must be simulated, or if a whole set of events must be divided across the worker threads, the retrieval cost is constant, as they are at the beginning of the timestamps list. As shown, the proposed scheduling algorithm produces no difference in the operations' costs depending on the number of worker threads availabe to follow through the simulation process.

Concerning point B), our scheduling algorithm divides the simulation's execution into several rounds, and in each round t all the k worker threads decide for a master thread via a specifically-targeted leader election procedure.

In particular, upon each iteration of the main loop, only the master thread executes the Events Assignment procedure, which splits (in case its cardinality is large enough) the first Event Set into k subsets, one for each worker thread. A synchronization barrier guarantees that if any worker thread is still processing events from round $t-1$, then no events assigned during the current round is processed. At the same time, this allows any thread which has finished processing events from round $t-1$ to try being elected as leader, and then starting the events assignment procedure.

A. Implementation of the Multi-Master Loop architecture within LTE-Sim

The original version of the LTE-Sim platform adopted a serial and linear event scheduler where events are organized into a linked list, ordered according to their timestamp, and executed once at a time during the simulation.

We extended the simulation framework by implementing the event management approach reported in Sec. III. To this end, we have completely re-implemented LTE-Sim's scheduling subsystem, relying (for portability reasons) on the Boost library [12] and the Standard Template Library (STL).

We have implemented the concurrent event queue presented in Fig. 3 using standard STL C++ containers: we modeled the *timestamps list* and the *concurrent events array* with the `std::list` and the `std::vector` containers, respectively. In this way we can reduce at the minimum the number of operations required to obtain a whole set of events by relying on `list::front()`, `list::pop_front()`, `vector::front()`, `vector::at()`, and `vector::erase()`) standard methods. Per-thread data have been implemented using boost's Thread Local Storage (TLS), namely `boost::thread_specific_ptr`.

The Multi-Master Loop uses `boost::barrier()` for handling thread synchronization and the `atomic_test_and_set` primitive for implementing the Leader Election procedure[2]. In particular, the only thread which, within a round, is able to successfully execute the `atomic_test_and_set` call on the multimaster guard is elected as leader, is charged of assigning events across all the available worker threads, and then remises the role by atomically resetting the multimaster guard.

B. Correctness of the Approach

The scheduling algorithm hereby presented allows dispatched simulation events which are marked with the same timestamp to be concurrently executed, independently of the generation order and/or the insertion order in the global concurrent event queue. Therefore the implemented concurrency control scheme maintains a high degree of parallelism by ensuring that: i) the read/write operations on simulation state performed by an executed event e on the simulation state S appear as they happened at same indivisible point in time associated with the logical simulation time T_e in which e has been processed; ii) all the executed events perform the same operations and produce the same outcome as they were processed sequentially without violating logical simulation time advancement. For this reason, if we model an event e's execution as an atomic transaction τ_e [13] to be considered committed whenever e is executed (i.e. it can be established a simulation time T^* such that each event e' executed at a time $T_{e'} < T^*$ is already committed and $T_e < T^*$), we can

[2]We note that although `atomic_test_and_set` is machine-dependent, it is available on most off-the-shelf platforms via, e.g., the `atomic.h` header, since it can be easily implemented on any modern architecture.

adopt the *serializability* consistency criteria [13], [14] over the histories of the committed events as the target correctness criteria of the proposed solution.

Before showing the proof we formalize the concepts of *history* on executed events and *operation*. A history H_{T^*} over a set E of committed events at simulation time T^* consists of i) a partial order of operations that reflect the read/write operations performed $\forall e \in E$ on the simulation state together with the *begin* (i.e., the invocation of e) and the *complete* (i.e., the commit of e), and ii) the version order $\ll$ that specifies a total order on the logical object's versions (i.e. variations on portions of the simulation states during the advancement of the logical simulation time) created by committed events. A $write$ operation on an object x issued by an event e is denoted by $w_e(x_e)$ while a $read$ operation on a version $x_{e'}$ of object x is denoted by $r_e(x_{e'})$.

Taken any two events e and e', the following dependencies might occur: i) e' directly *read-depends* on e if there exists an object x such that e' executes a read $r(x_e)$; ii) e' directly *write-depends* on e if there exists an object x such that e executes a write $w(x_e)$, e' executes a write $w(x_{e'})$ and $x_{e'}$ immediately follows x_e in the total order defined by $\ll$ on x; By Property 1 and 2, we know that taken a Concurrent Event Set C associated with a timestamp T_C in the Concurrent Event Queue, $\forall e, e' \in C$ neither dependency i) nor ii) can occur.

We can therefore build a Direct Serialization Graph $DSG(H_{T^*}, \ll)$ over a history H_{T^*} as stated in [14] in order to define serializability in terms of topological properties on that graph. In particular a graph $DSG(H_{T^*}, \ll)$ contains a node N_e for each executed event e in H_{T_C} and a directed edge $N_e \rightarrow N_{e'}$ for each pair of committed events e, e' in H_{T^*} such that either dependency i) or ii) occurs. Then a history H_{T^*} is serializable if the associated $DSG(H_{T^*}, \ll)$ does not contain oriented cycles as defined in [13].

Therefore the correctness proof of SSMS is formalized in the following Theorem:

Theorem 1. *For each simulation time value T^* and for each history H_{T^*} of committed events admitted by the scheduling algorithm, then the $DSG(H_{T^*}, \ll)$ graph does not contain any oriented cycle.*

Proof: We prove that the $DSG(H_{T^*}, \ll)$ does not contain any oriented cycle by showing that for each edge $N_e \rightarrow N_{e'}$, $T_e < T_{e'}$ always holds.

If an edge $N_e \rightarrow N_{e'}$ is in $DSG(H_{T^*}, \ll)$ we have to distinguish two cases:

1) e' directly *read-depends* on e. In this case the event has performed a read operation on an object x by accessing logical the version x_e having the greatest timestamp T_e less than $T_{e'}$. Therefore $T_e < T_{e'}$.
2) e' directly *write-depends* on e. e' overwrites a value of an object x already written by e. This is admitted only if $T_e < T_{e'}$.

■

By Theorem 1 follows that every committed history generated by our scheduling algorithm guarantees serializability.

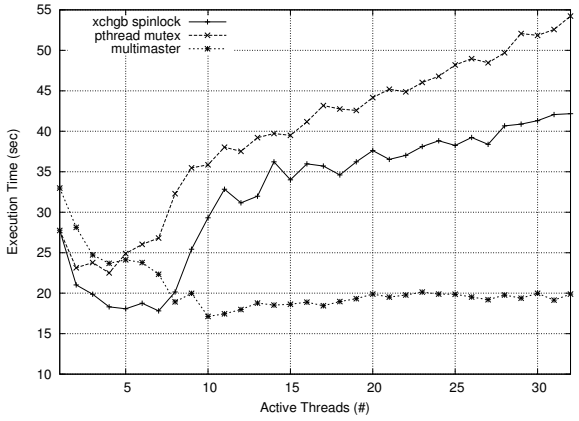

Fig. 4. Comparizon of synchronization cost in a SingleCell Scenario

IV. Experimental Results

In order to evaluate the performance gain achievable by the adoption of the presented Multi-Master architecture, we have analyzed both *SingleCell* and *MultiCell* scenarios. The *SingleCell* scenario is composed by only one LTE cell, one eNB and 50 users. Each user moves into the network at 3 Km/h and receives a downlink traffic modeled through the infinite-buffer application. The eNB distributes radio resources among mobile terminals by means of the PF scheduler. The *MultiCell* scenario is composed by 19 cells and a number of users distributed within the entire network. Similar to the previous scenario, also in this case each user moves at 3 Km/h and receives a downlink traffic modeled through the infinite-buffer application, and the eNB distributes radio resources among mobile terminals by means of the PF scheduler. All conducted simulations last 60 simulated seconds each.

The hardware architecture used for testing our proposal is a 64-bit NUMA machine, namely an HP Proliant server, equipped with four 2GHz AMD Opteron 6128 processors and 64GB of RAM. Each processor has 8 CPU-cores (for a total of 32 CPU-cores) that share a 10MB L3 cache (5118KB per each 4-cores set), and each core has a 512KB private L2 cache. The operating system is 64-bit Debian 6, with Linux kernel version 2.6.32.5.

All presented results have been obtained by averaging 5 different simulation runs' outcomes, and time measures have been taken via the standard `gettimeofday()` service, offering microsecond granularity. The intrusiveness of this approach is negligible given an overhead of less than 1 ms on current conventional machines for the couple of calls required to take start and end time of the interval defining the latency sample to be evaluated.

The first analysis we propose would demonstrate the effectiveness of the proposed approach compared with respect to other classical symmetric strategies (see Fig. 2). To this end, we implemented two types of symmetric schemes. The former ensures synchronization among threads using the standard `xchgb` assembly instruction on Intel-compliant x86 architectures. The latter, instead, uses pthread mutexes to perform

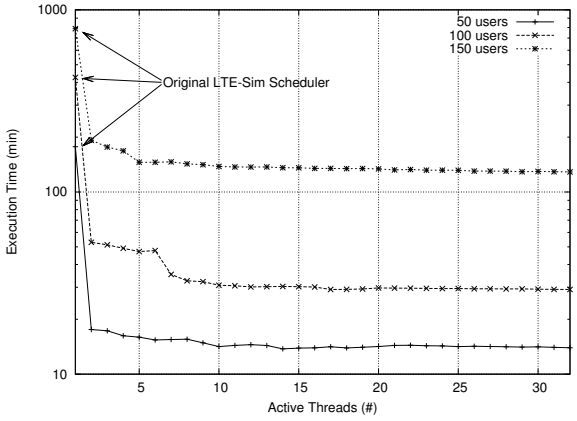

Fig. 5. Multi-Master's Total Execution Time in a MultiCell Scenario

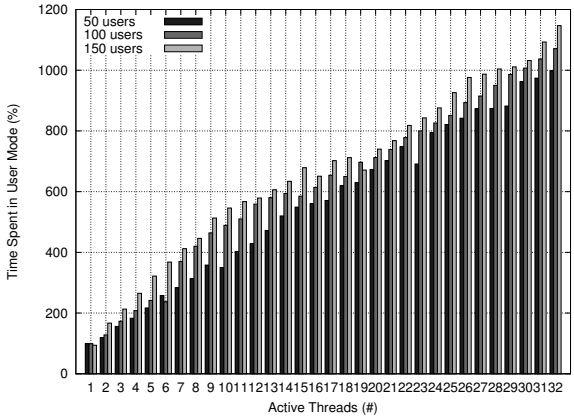

Fig. 6. Multi-Master's CPU Utilization in a MultiCell scenario

the same task. The *SingleCell* scenario has been considered in the comparison. Such scenario allows to measure which is the actual overhead induced by a parallelized scheduling algorithm, since it shows a reduced degree of parallelism (i.e. the number of concurrent events is very limited).

In Fig. 4, we report the simulation execution time by varying the number of active worker threads in the system. From the results it is evident that symmetric schemes reach a thrashing point with a very low number of worker threads, due to the high logical contention on the event queue. A different behavior has been registered for the Multi-Master scheme that always guarantees an evident reduction of the simulation time with respect to the other approaches and to the single-tread implementation. Furthermore, we can observe that the proposed scheme achieves best performance with 10 worker threads and then mostly stalls.

To provide a further insight, we evaluate the performance of the proposed Multi-Master scheduler in a more complex scenario (i.e., the *MultiCell*) considering different number of mobile users. The execution time of the simulation is reported in Fig. 5. We can observe that, despite the high number of parallel events generated during the simulation, the performance gain achieved by the presented solution is fully able to exploit the potential of multi-processing architectures, thus reaching a maximum speedup in the order of 90%, 90%, and 80% when the number of users are set to 50, 100, and 150, respectively. It is interesting to additionally note that, when a high number of concurrent events is generated by the simulation scenario, the scheduler incurs in no thrashing at all.

Finally, we report in Fig. 6 the the CPU utilization, showing that it linearly grows with the number of active worker threads in the system.

V. Conclusion

In this work we have presented the new Multi-Master scheduling algorithm for the LTE-Sim simulation package. We have presented design indications for our proposal, which allow any LTE simulation engine to implement its own version of the algorithm, and we have presented a proof showing the correctness of our proposal. We have additionally assessed the vailidity of our proposal relying on different simulation scenarios provided by the LTE-Sim simulation package.

References

[1] Cisco, *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2010-2016*.

[2] E. Dahlman, S. Parkvall, and J. Skold, *4G LTE/LTE-Advanced for Mobile Broadband*. Academic Press, 2011.

[3] N. Baldo, M. Miozzo, M. Requena-Esteso, and J. Nin-Guerrero, "An open source product-oriented LTE network simulator based on ns-3," in *Proceedings of the 14th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems*, ser. MSWiM. ACM, 2011, pp. 293–298.

[4] G. Piro, L. A. Grieco, G. Boggia, F. Capozzi, and P. Camarda, "Simulating LTE cellular systems: An open-source framework," *IEEE Transactions on Vehicular Technology*, vol. 60, no. 2, pp. 498–513, feb 2011.

[5] F. Capozzi, G. Piro, L. A. Grieco, G. Boggia, and P. Camarda, "A system-level simulation framework for LTE femtocell," in *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, ser. SIMUTools. ICST, mar 2012.

[6] F. Capozzi, G. Piro, L. Alfredo Grieco, G. Boggia, and P. Camarda, "On accurate simulations of lte femtocells using an open source simulator," *EURASIP Journal on Wireless Communications and Networking*, 2012.

[7] F. Capozzi, G. Piro, L. A. Grieco, G. Boggia, and P. Camarda, "Downlink packet scheduling in lte cellular networks: Key design issues and a survey," *IEEE Commun. Surveys and Tutorials*, 2012.

[8] R. Vitali, A. Pellegrini, and F. Quaglia, "Towards symmetric multi-threaded optimistic simulation kernels," in *Proceedings of the 26th International Workshop on Principles of Advanced and Distributed Simulation*, ser. PADS. IEEE Computer Society, Aug. 2012, pp. 211–220.

[9] ——, "A load-sharing architecture for high performance optimistic simulations on multi-core machines," in *Proceedings of the 19th IEEE International Conference on High Performance Computing*, ser. HiPC. IEEE Computer Society, dec 2012.

[10] A. Pellegrini, R. Vitali, and F. Quaglia, "The ROme OpTimistic Simulator: Core internals and programming model," in *Proceedings of the 4th ICST Conference of Simulation Tools and Techniques*, ser. SIMUTools. ICST, 2011.

[11] B. Karlsson, *Beyond the C++ Standard Library: An Introduction to Boost*. Addison Wesley Professional, aug 2005.

[12] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency control and recovery in database systems*. Addison-Wesley Longman Publishing Co., Inc., 1986.

[13] A. Adya, "Weak Consistency: A Generalized Theory and Optimistic Implementations for Distributed Transactions," Massachusetts Institute of Technology, Cambridge, MA, USA, Tech. Rep., 1999.