Investigation of Containerizing Distributed Petri Net Simulations

Jan Henrik Röwekamp, Daniel Moldt, and Matthias Feldmann

University of Hamburg, Vogt-Kölln-Straße 30, 22527 Germany
http://www.paose.de

Abstract. Reference nets combine the power of an object-oriented programming language with the formalism of Petri nets. Using them makes it possible to model complex concurrent systems easier. By doing so with a tool like Renew, we achieve a directly executable (local) simulation of the system. In recent publications, ideas have been presented to transpose these simulations into one distributed simulation. However, being a genuinely concurrent system, the problem arises, that some nodes are less utilized than others slowing down the simulation in its entirety. In this paper, we define requirements to a distributed simulation and evaluate how specific technical approaches harmonize with them. We present a first - reference net based - prototype for an approach using containerization and Docker for the execution of a single net simulation truly distributed on several (virtual) machines. As an example of a workload with differing complexity, the prototype calculates parts of the Mandelbrot set in a distributed manner. A significant advantage of the approach is that inherent complexity of a problem may be unknown without restraining the method.

Keywords: High-level Petri nets · Reference nets · Distributed Simulation · Docker.

1 Introduction

Reference nets are a high-level Petri net formalism and provide a strong object-oriented foundation to model complex systems. Because reference nets follow the principal of nets within nets, use net instances (similar to object orientation) and are organized hierarchically, communication between nets imposes a certain overhead. Synchronous channels are a practical solution in local cases. However to find a binding unification is required, which gets complicated in the context of synchronous channels in the distributed case. A solution using very simple synchronous channels has been proposed [13]. Later we have shown a simple prototype using these simple channels in combination with virtual machines [10]. In this paper, we discuss the requirements for a distributed simulation and present a simple Docker-based prototype of a distributed simulation.

Motivation

When simulating reference nets, it is desirable to not only simulate a distributed system but to distribute the simulation altogether. Having a simulation infrastructure that copes with different complications inherent to a distributed system, gives the possibility to achieve this goal. Using a distributed net simulation has several advantages, for example, it benefits all parts of the software engineering life cycle. It can be used to model prototypes and mock-ups during requirements engineering, implement the actual system during implementation or help with verification during maintenance.

2 Basics, Related Work, and Technical Background

2.1 Net Basics and Tools

Some Petri Net basics of reference nets and their toolset are explained, as they form the basis for our conceptual and technical proposal.

Nets within Nets and Synchronous Channels Petri nets are usually considered as monolithic structures. High-level Petri nets introduced hierarchies and allowed splitting of nets into subcomponents of the whole structure. Binding of subnets with the embedding nets has been discussed intensively.

Reference nets [7] used synchronous channels [5] for the integration of Object Petri nets [14] and Object-oriented Petri Nets [8]. Synchronous channels allow an active coupling of net instances at runtime. The binding ensures that all variables of all involved transitions are matched entirely before they are executed atomically. As mentioned above the binding is a highly sophisticated operation. A distributed synchronization across different physical machines is not feasible. Therefore we introduced a simpler version of the synchronous channels [13].

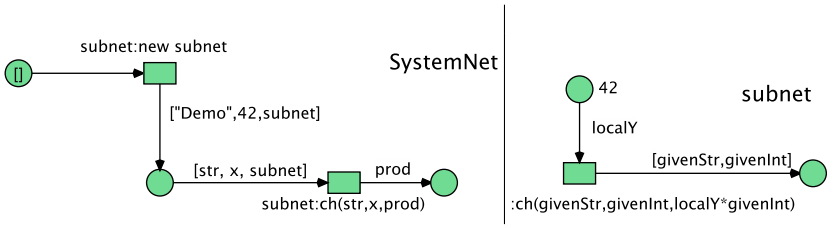

Fig. 1. `System net` example and `Object net` example with synchronous channel `ch`

In Figure 1 the left net is the `System net` called `SystemNet` and other net is the `Object net` called `subnet`. An instance of the `SystemNet` net template is instantiated, fires the first transition and generates a token that contains a string "Demo", an integer 42 and the reference to the generated `Object net` instance of a `subnet` net template called `subnet`. Binding of the two transitions via the

channel `ch` is first collecting the possible bindings in the `SystemNet`, binding it to the parameters of the channel (`str` will be bound to "Demo", `x` to 42 and `subnet` to the net instance reference in this example). Subsequently, the binding of the nonbound parameters attached to the transition in the subnet is calculated (a possible binding in the example would be `localY` to 42, `givenStr` to the same value as `str` and by that "Demo", and `givenInt` to the value of `x`, which is also 42). Then the firing of the transitions in both nets occurs concurrently. Resulting in the marking of `1764` (the value, that will be bound to `prod`) in the right most place in the `SystemNet` and the tuple `["Demo,42"]` in the right most place in the `Object net`. Please note the asymmetric call: the caller (`SystemNet`) has to have the reference of the callee (`subnet`) to invoke the channel `ch` with `subnet:ch(str,x,prod)`.

Renew For reference nets, the tool Renew has been developed. While it can be used for many other formalisms reference nets and its channels are the primary formalism. It allows the modeling and the execution of reference nets in a genuinely concurrent mode.

In [12] a prototype for distributed simulation has been introduced. It uses RMI (Remote Method Invocation) as the technical implementation of communication. Synchronous channels are implemented with a two-way synchronization: Transitions of a net instance (caller) with special inscriptions call a remote transition (callee). The caller proposes a binding for all parameters of the synchronous channel for which it can provide some values to the callee. The callee then acts as a caller for other synchronous channels attached to it. The callee calculates the missing values of the parameters of its caller (with the help of its callees) and reports the values back to its caller. The caller then decides if all involved transitions that contributed to the calculation of the binding will fire atomically. In that case, it initiates the firing, which is performed in all involved simulators.

Other formalisms than the reference nets of Renew allow slightly modified channel inscriptions. However, they provide only synchronous channels for local calls of other nets. Communication with other machines is only realized via other frameworks like RMI, ActiveMQ, and others.

2.2 Containers vs. Virtual Machines

Virtual machines are designed to run applications on a particular system. A hypervisor is located on top of the infrastructure and creates and runs virtual machines [3]. Each virtual machine runs its unique operating system, allowing different operating systems to be run on the same infrastructure. Since each virtual machine has its own binaries, libraries, and applications, virtual machines may use multiple gigabytes of size [3, 11].

Divergent to virtual machines, containers are located on top of the infrastructure and its host OS. The containers share the OS Kernel and may share the binaries and libraries [2]. These shared components are read-only to the containers [3]. Therefore containers are lightweight and may use only a few megabytes in

size, allowing them to start faster [2]. By sharing the components, only a single OS is necessary, and by this, the management overhead is reduced [3].

Docker Docker is an open source container virtualization technology [1]. Docker allows reusing components over multiple containers, by using components of their base image (which can also have a base image) and by this minifying the size and the start-up time of containers [1, 2]. By assigning a different namespace to each container, docker prevents containers from having access to anything other than what was defined when creating the container [1].

2.3 Container management

Container management is responsible for all actions concerning containers [6]. Management software can be used to simplify the administration of containers and help to set the access of containers [2]. Automation can be used to configure and start containers depending on the requirements [9].

Autoscaling Multiple containers of the same image can be used to split a load across multiple containers. Autoscaling is responsible for adjusting the number of containers to meet the demand [9].

2.4 Requirements

For further evaluation, it is desirable to outline our requirements for a distributed simulation. To run a distributed simulation several (technical) aspects need to be considered. The three main categories of interest are: The net formalism, the infrastructure and finally the engine used to run the simulation. We assume that all these are possibly alterable. As a distributed simulation itself is also a distributed system, well-known requirements for distributed systems apply as well. Therefore, resources are a crucial term in all three categories. A distributed simulation should keep the resource demand as low as possible. Also, derived from distributed systems, security concerns are to be considered. This is mainly an infrastructural topic, but to a degree applies to the engine itself and the net formalism as well. In our context security is a generalized term on how good components of the net are protected from unauthorized access, regardless of it originating from within or without the simulation itself. Finally, a net formalism related requirement is splitability and distributability of net parts. While in an ideal world every concurrent transition firing could run distributed as well, this is very hard to achieve due to synchronization constraints. However, achieving a good distributability is one major goal in designing a distributed simulation environment.

3 Distribution of the Simulation

Partitioning of the simulation is crucial for a distributed simulation. Out of the net formalisms we choose reference nets, because of their net instance capabilities. Similar to object-oriented programming nets are like classes, of which net instances can be created. Net instances themselves can be treated as a token in a place in another net instance. Being able to create net instances on demand results in significantly increased flexibility. However, there is a drawback: The resource demand of the simulation may increase during running time. This also means that the simulation environment might not be known in its entirety when starting the simulation. To be able to dynamically provide more computation power, in case the simulation is already fully utilizing the underlying system, we investigate three kinds of infrastructure and their capabilities of extending the simulation on demand. These three main approaches are bare metal machines, virtual machines, and containers.

3.1 Simulation Extension using Bare Metal Machines

The classic approach is to start one part of the simulation directly on a physical machine. This method has the least overhead but is also somewhat static. Adding an additional unit requires a fully installed machine, that is connected to the network. The simulation also needs to wait until the new machine is booted before it can be used. There is also a significant installation overhead to be considered, as all components need to be installed before the first run.

3.2 Simulation Extension using Multiple Virtual Machines

The next logical step is to migrate the distributed simulation into virtual machines as we have proposed earlier [10]. Using virtual machines renders parts of the simulation transportable, which reduces dependency on the infrastructure. However, virtual machines are large, and the aspect of delayed integration into the running simulation remains, as the virtual machines also have to boot.

3.3 Simulation Extension using Containers

The main shortcomings of the virtual machine approach are size and boot time. The usage of containers can elegantly address both. As mentioned earlier Docker as a container virtualization technology comes with integrated security features despite sharing resources on a machine. Using containers finally allows for portable, isolated simulation parts, that can be added to the simulation promptly.

4 Prototype

Per assumption (see section 2.4) net formalism, infrastructure and engine may be altered. Using Renew (with Distribute plugin) we can apply changes to the base

formalism of reference nets and also extend the engine using plugins. Distributed synchronous channels are an example of an element diverging from the base reference net formalism. As this work examines the use of containers a possibly changing underlying infrastructure is of primary interest. We present a brief prototype to show the general extendability of a simulation using Renew with Distribute plugin. Extendability is a necessary precondition to be able to adapt to changing infrastructure. Renew is technically a Java extension. However, it provides true concurrency, non-determinism and conflict management (fairness). Also note, that the Distribute plugin is based on Java RMI, but completely abstracts this fact from the modeler.

4.1 Concept

A central work distributor will be used to offer work packages. Other parts of the simulation can synchronize with it to fetch packages and deliver done work. These other parts are realized within prepared (container-)images, so by merely launching such an image into a container, the simulation gets extended. To realize a proper utilization of the system, partitioning the workload is desirable in such a manner every simulation node is utilized. Note, that we use a centralized approach (instead of, e.g., work stealing) to avoid the necessity for each (maybe newly initialized) worker to know each other currently active worker.

4.2 Hypothesis

With the proposed prototype we mainly want to evaluate the hypothesis, whether it is possible to dynamically extend a running reference net simulation during runtime using containerization technology. As a secondary goal, a high system load is desirable to avoid inefficient utilization of resources.

4.3 Technical Realization

The concept is realized using Docker as container technology, Renew 2.5 as simulator using the 'Distribute' plugin [13] running on OpenJDK 1.8. Reference nets with Distribution extensions are used as underlying formalism. For demonstration purposes, the simulation runs on six different Linux Mint 18.3 virtual machines using Oracle Virtualbox 5.2.8 on top of a Windows 10 system with an Intel Core i7-6800K CPU and 64GB of RAM. Each VM was set up with 10 GB of ram and a single CPU core. Please note, that the realization with nested containers inside virtual machines is due to simplicity to set up, isolation from other processes and uniformity of the used machines. The proposed solution is to use containers as mean to extend the simulation *instead* of virtual machines. One VM is used to run the distributor, one VM runs the central registry, and the remaining four VMs each run a net, that fires up docker containers containing additional parts of the simulation.

4.4 Application

There are several possible application scenarios, like modeling of a complex system, multi agent applications, software verification and more. However, all of these are rather complex and since this prototypes intended use is to evaluate, whether a simulation is dynamically extendable with containers, a more straightforward task was chosen. The computation and visualization of the Mandelbrot set [4] serves as a good vehicle to acquire readily splittable workloads, as well as different workload severities to potentially observe load balance problems. It is also an easy enough problem to fit this prototype.

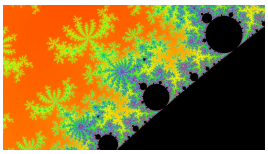

Fig. 2. Visualization of the Mandelbrot set

The Mandelbrot Set Given $z_0 \in \mathbb{C}$ and $z_{i+1} = z_i^2 + z_0$, the Mandelbrot set is defined as $M = \{z_0 \in \mathbb{C} | lim_{n\to\infty} z_n < \infty\}$. Projecting the n from which onwards the progression diverges onto the color spectrum yields the well-known visualization of the Mandelbrot set depicted in figure 2. Given a maximum number of iterations to evaluate, note that the black areas are very computation intensive, as the computation cannot be aborted early.

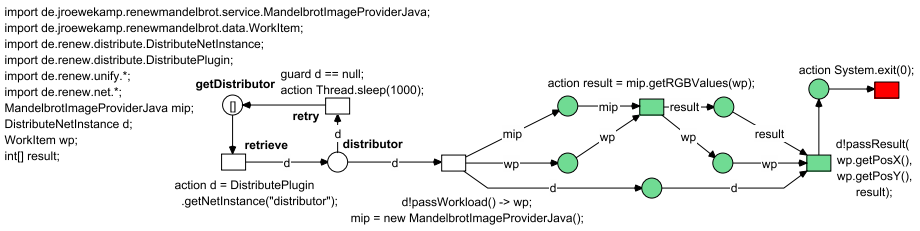

Fig. 3. Mandelbrot Worker Net. Launched in a Docker container.

To demonstrate the difference between the dynamic distribution of work packages and the static distribution onto a fixed number of instances, the image is computed twice in two alternative ways: First, the image is split into four equal parts. Four workers are launched and each worker processes its equal sized part of the problem. Second, the image is split into 16 parts. These are dynamically

assigned to newly created containerized simulation parts (workers). By doing so the second alternative automatically adds computation agents to the simulation, which is a very basic form of autoscaling (see section subsec:containerMgmt) without dynamic load adaption. Both alternatives are calculated with the same nets, only the configuration is different to implement these two approaches. The blueprint for a worker net can be found in figure 3 and the distributor net in figure 5. A launcher net starts new net parts (docker containers). The launcher is shown in figure 4. Please note that the transition inscription for the channel differs from the channel explained in Figure 1: `d!passWorkload() -> wp`. The syntax is part of the distributed reference net formalism and required for the binding search, but are too complicated to elaborate on here. As for intuition imagine `->` as output parameter of a firing, `<-` as input parameter and `!` as synchronization with a non-local net instance. For further details see [13].

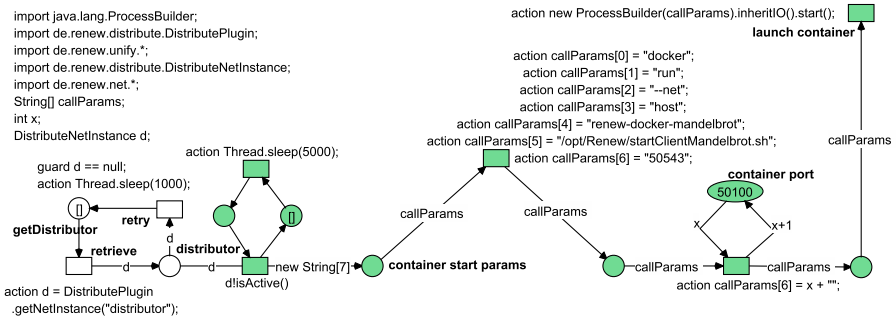

Fig. 4. Mandelbrot Local Launcher Net

4.5 Evaluation

Overall the Docker-based simulation ran very well. Additional simulation parts were launched as expected to extend the simulation accordingly. The image in figure 2 was computed in all tests using a resolution of 1920x1080 pixels and an iteration depth of 100.000. The first test with equal deterministic workload distribution took an average of 150 seconds to complete, while the second test with dynamic workload distribution took an average of 107 seconds. Despite the overhead of launching new simulations in the second test the approach using fixed workload distribution took about 40% more time in this specific example to complete. The overall observed system load was higher using the dynamic distribution, as expected. These numbers are the average of about fifteen independent runs, but each diverging only by 3-5 seconds from the average.

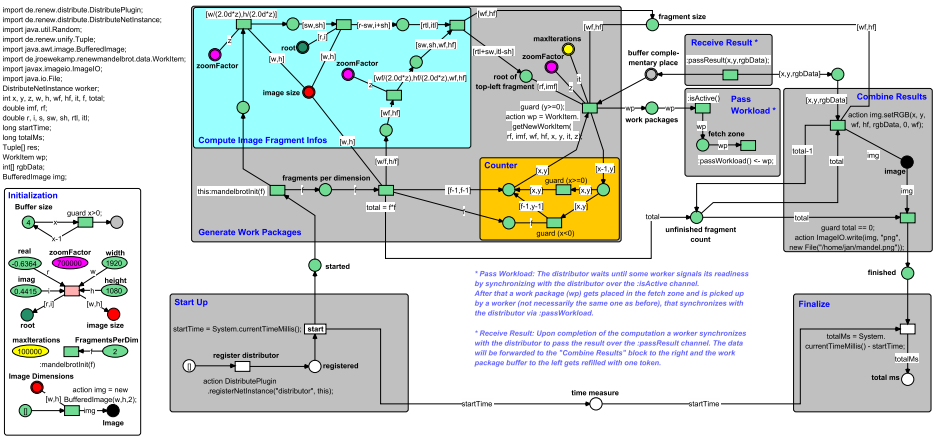

Fig. 5. Mandelbrot Distributor Net

5 Conclusion

In this paper requirements on a distributed Petri net simulation have been presented. This distributed simulation is performed for a single net system using the distributed resources to form a single simulation.

Different approaches using bare metal machines, virtual machines and containerization have been discussed. Finally, a simple prototype using the containerization technology has been shown.

Future work can include but is not limited to fully fledged reference net based autoscaling, net partitionability considerations, and evaluation of more complex net tasks.

References

1. Anderson, C.: Docker. IEEE Software **32**(3), 102–105 (2015). https://doi.org/10.1109/MS.2015.62, https://doi.org/10.1109/MS.2015.62
2. Bernstein, D.: Containers and cloud: From LXC to Docker to Kubernetes. IEEE Cloud Computing **1**(3), 81–84 (2014). https://doi.org/10.1109/MCC.2014.51, https://doi.org/10.1109/MCC.2014.51
3. Brey, P.: Containers vs. Virtual Machines (VMs): What's the Difference? | NetApp Blog (Mar 2016), https://blog.netapp.com/blogs/containers-vs-vms, last accessed 22 Apr 2018
4. Brooks, R., Matelski, J.P.: The dynamics of 2-generator subgroups of PSL (2, C). In: Riemann surfaces and related topics: Proceedings of the 1978 Stony Brook Conference, Ann. of Math. Stud. vol. 97, pp. 65–71 (1981)
5. Christensen, S., Hansen, N.D.: Coloured Petri nets extended with channels for synchronous communication. In: Valette, R. (ed.) Application and Theory of Petri

Nets 1994, 15th International Conference, Zaragoza, Spain, June 20-24, 1994, Proceedings. Lecture Notes in Computer Science, vol. 815, pp. 159–178. Springer (1994). https://doi.org/10.1007/3-540-58152-9_10, http://dx.doi.org/10.1007/3-540-58152-9_10

6. Inagaki, T., Ueda, Y., Ohara, M.: Container management as emerging workload for operating systems. In: 2016 IEEE International Symposium on Workload Characterization (IISWC). pp. 1–10 (Sept 2016). https://doi.org/10.1109/IISWC.2016.7581267
7. Kummer, O.: Referenznetze. Logos Verlag, Berlin (2002), http://www.logos-verlag.de/cgi-bin/engbuchmid?isbn=0035&lng=eng&id=
8. Maier, C., Moldt, D.: Object coloured Petri nets – A formal technique for object oriented modelling. In: Agha, G., De Cindio, F., Rozenberg, G. (eds.) Advances in Petri Nets: Concurrent Object-Oriented Programming and Petri Nets, Lecture Notes in Computer Science, vol. 2001, pp. 406–427. Springer-Verlag (2001)
9. Netto, M.A.S., Cardonha, C., Cunha, R.L.F., Assuncao, M.D.: Evaluating auto-scaling strategies for cloud computing environments. In: 2014 IEEE 22nd International Symposium on Modelling, Analysis Simulation of Computer and Telecommunication Systems. pp. 187–196 (Sept 2014). https://doi.org/10.1109/MASCOTS.2014.32
10. Röwekamp, J.H., Moldt, D., Simon, M.: A simple prototype of distributed execution of reference nets based on virtual machines. In: Proceedings of the Algorithms and Tools for Petri Nets (AWPN) Workshop 2017. pp. 51–57 (Oct 2017)
11. Salah, T., Zemerly, M.J., Yeun, C.Y., Al-Qutayri, M., Al-Hammadi, Y.: Performance comparison between container-based and vm-based services. In: 2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN). pp. 185–190 (March 2017). https://doi.org/10.1109/ICIN.2017.7899408
12. Simon, M.: Concept and Implementation of Distributed Simulations in Renew. Bachelor thesis, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg (Mar 2014)
13. Simon, M., Moldt, D.: Extending Renew's algorithms for distributed simulation. In: Cabac, L., Kristensen, L.M., Rölke, H. (eds.) Petri Nets and Software Engineering. International Workshop, PNSE'16, Toruń, Poland, June 20-21, 2016. Proceedings. CEUR Workshop Proceedings, vol. 1591, pp. 173–192. CEUR-WS.org (2016), http://CEUR-WS.org/Vol-1591/
14. Valk, R.: Petri nets as token objects - an introduction to elementary object nets. In: Desel, J., Silva, M. (eds.) 19th International Conference on Application and Theory of Petri nets, Lisbon, Portugal. pp. 1–25. No. 1420 in Lecture Notes in Computer Science, Springer-Verlag, Berlin Heidelberg New York (1998). https://doi.org/10.1007/978-3-540-27793-4_29, http://dx.doi.org/10.1007/978-3-540-27793-4_29