Adaptation of System Dynamics Model Execution Algorithms for Cloud-based Environment

Alexey Mulyukin, Ivan Perl
ITMO University
Saint-Petersburg, Russia
alexprey@yandex.ru, ivan.perl@corp.ifmo.ru

***Abstract*—This paper presents a process of adaptation of system dynamics models execution algorithms to cloud-based environment. System dynamics is an aspect of systems theory as a method to understand the dynamic behaviour of complex systems. Existing modeling algorithms used in popular modeling solutions are either not available for free use or have several disadvantages which prevent them from being used in distributed cloud environment. Adaptation of execution algorithms aimed not only to adapt execution process to distributed parallel environments with higher reliability and wider range of possible applications, but also to improve system dynamics model execution performance. For example, existing algorithms of model execution which are not ready for distributed environments will fail to complete modeling task in case of hardware failure, and optimized ones are able to smoothly transfer execution process from one node to another with minimal impact on overall model execution progress. Such capabilities help to save many resources and, especially, time on execution re-runs. In this paper described algorithms and approaches designed for sdCloud solution which are focused on transferring execution of system dynamics models into distributed cloud-based environment and shown extra benefits brought to modeling process by shift to the cloud.**

I. Introduction

System dynamics is an aspect of systems theory which is an approach to understand the dynamic behavior of complex systems. The system dynamic models consist of stocks and flows. The stocks in scope of system dynamic represent some real values of our world that can be changed in over time. The flows in scope of system dynamic represent a function describing stocks values changes. Those simple elements allow constructing models of any complexity level. Such models can describe real world processes or systems in required scale for specific research needs. The system dynamic models cover many areas of our world including but not limited to economic, medicine, social, mechanic and engineering. In scope of system dynamics used the system of linear equations that solved by iterative approach in required model time space range that defined by researches that investigate model behavior. The model solving process is also can be called as model execution process. When we are talking about execution of a system dynamics model we are assuming a process of sequential computation of model states over a given period with the provided modeling step, representing a minimal time frame to navigate in modeling results [1]. For example, we can take the simple epidemic model, that describe how disease spreads among area population, and this model built for answering the next question: "Can disease spreads to all population or disease is disappeared and all population will be healthy?" and answering to this question called is model execution [1], [2].

II. Overview of exists solutions

To perform model execution on cloud computing cluster we should consider existing model execution algorithms used in System Dynamics. As an example, we can consider the next modern solutions for computing system dynamic models: PySD, SDEverywhere, sd.js, Vensim, AnyLogic and AnyLogic Cloud. We split those solutions to three categories and dig into each deeper with detailed overview of each category.

A. Open source libraries

PySD, SDEverywhere and sd.js are open source libraries that implements execution engines for computing system dynamic models in xmile or vensim formats, those formats are most popular among modelers for now [3], [4], [5]. Those solutions based on linear computation process, that just takes input configurations, process it on single CPU core and in local memory space and then returns results when all computations are completed. The one of key issue of those solutions are targeting software engineers more than modelers and scientists. Those libraries can be easy integrated in your applications, but it hard to use for end users. To compute a simple model, you should have basic knowledge of programming languages. For example, if you want to compute model via SDEverywhere you should install on your local machine compiler of C language and many other dependencies which are required for this library, after that use UNIX console and generate C code of your input system dynamics model, compile them, run and after it you can see results in console or raw output file. And it will be just raw output data which is not interested, and this output should be "manually" handled. However, if we have skills in programming you can integrate those solutions and make automatic analytics of output or render need visualizations. So, those approaches make a high adaptation threshold and correspondingly have a small number of end-users. Meanwhile, those solutions are very interesting to community, because supported by open-source community and provided by free license.

B. Proprietary desktop solutions

The second group of tools consists of proprietary solutions, like Vensim and AnyLogic, which are very popular for now and have many active users [6], [7]. Let's look closer at how those applications are work. We can't give accurate assessment of used algorithms, but we can say why open-source solutions has more prospects in development. Vensim and AnyLogic solutions have several differences in comparison with open-source solutions. For example, Vensim and AnyLogic have changes in build-in algorithms of model computing that can provide events from computation process with information about changes to UI elements. So, it makes results of model executing animated and easy to understand for end-user. However, those solutions have issue related to computing system dynamic models on local environment without possibility to attach additional computation nodes to improve performance and reliability. Also, those solutions can't be used as a part of your business applications, you can't integrate those solutions by easiest way.

C. Web-oriented solutions

And to the last we look at web-oriented solutions for computing system dynamic models, like: sdCloud, AnyLogic Cloud, Insight Maker [7], [8], [9], [10]. All those solutions are available for any user that have connection to Internet and have installed an any Internet browser. This provides a low adaptation threshold for end-users to use any of those web-applications. However, each of those applications implemented in different ways and provide different features to end-user. Looks detailed for each one.

Firstly, look at Insight Maker, this solution provides to user possibility to build models directly in browser with WYSIWYG editor. Also, this web-application allow saving models in cloud storage, execute and see results data visualization directly in browser. You can embed your model to any other website via embedding by iframe HTML tag as well. However, computation process with visualizations executed on local user machine (directly in user web browser). So, we can say that Insight Maker is not use all advantages of cloud-oriented solutions. For executing complex system dynamics models you still have a high-performance local machine. Using this solution, you can't export results in any formats like csv, for continuing work on best tools for analytics like Microsoft Excel, etc.

Now look at AnyLogic Cloud solution. This web-application provides to user possibility to execute models on internal cloud environment and views results visualization. However, user can't create model or create results visual representation directly in AnyLogic Cloud. To perform those actions, you should use desktop application and export your model to AnyLogic Cloud. Also, we should mention that all results are sent to end user as events via network connection and those events handled by browser scripts. Because all events provided for each time-step of model executing, this leads to high network abusing and high CPU usage. And those metric values are grown in proportion to complexity of visualization data of model executing. However, AnyLogic Cloud have interesting features with model input parameters optimization tasks.

And in the end, we look at sdCloud solution. This web-application provide to user possibility to execute models in most popular formats among modelers using internal cloud environment. sdCloud – it a cloud platform that wrap open source tools for working with system dynamic models directly in web browser. For now, this platform gives to user possibility to upload model definition, run it, and see or download results in any handful formats: table representation, programming representation (e.g. json, csv). So, user can use all advantages of open-source solutions, for using it we should just have any web-browser, even any simple mobile browser, and connection to Internet. However, issues with monolithic structure and algorithms there are still, because internally used the same approaches.

D. Generalizing

Looks backward to all of what we say in this section, we can extract key features of each kind of system dynamic model execution solutions and present differences in table representation in Table I.

TABLE I. COMPARING FEATURES FOR DIFFERENT KIND SOLUTIONS

Feature	Open-source	Proprietary desktop	Web-oriented (w\o sdCloud)
Supported by community	Yes	No	No
WYSIWYG model editor	No	Yes	Yes / No
Results visualization	No	Yes	Yes
User adaptation threshold	High	Middle	Low
Integration with business applications	Yes	No	No
High usage of local machine resources	Yes	Yes	Yes

So, we can extract all key issues that inherent to each kind of model execution applications. First issue is intense usage a local computation resources and this is a key and principal issue. And the second issue – high threshold of user adaptation for using and integrating those approaches to business applications.

Execution of system dynamic models requires many computing resources (i.e. processor time and memory) and existing algorithms are not designed for execution in cloud environment. The monolithic structure of used model computation algorithms is a general cause of this issue. By this cause, we can't split and use this approaches on few execution nodes in order to improve performance and reliability of built solution. There are some cases like power outage issue, network problems, or any other hardware or software faults. If model execution requires much time (for example 4-6 hours) and computing hardware or software will break on last steps of model execution, then all results will be lost, and model should be executed from beginning. This case is not usual, but it's possible. And if we want to build the cloud-oriented application, we should build the reliable solution that can

guarantee high availability of service to end-user. And when we talk about single model execution, described situations with issues of such kind is very rare and can't significantly affect a single user. However, when we talk about cloud-oriented solution of system dynamics model execution, we should provide the availability in long-term, for example weeks, months, years, etc. And in this period probability of this issue is more often than when we talk about single model run that can be proceed in half-day in average. We also need to consider continues modeling which require high available service for providing continues results to end-user [11].

The second issue of existing execution approaches is results data persistence. Model execution generates big data volumes for complex models over modeling time. And existing algorithms store these results in memory and only after fully completion of model execution sends results to external storages. This makes a load not just for memory, but also on network when this data volumes are sends to data storage.

So, our global goal is to design architecture for model execution process that can be used on cloud computing cluster and solves described issues reliably.

III. Goals

Based on previous section of this paper we can determine our step-goals which can help us to reach the global goal – build the cloud computing cluster for system dynamic model executing. We need to find a middle ground between ease of use by end users and easy of integration in external business applications. We should strive to quick provide of model results in handful formats and straightforward way for researches. Our sdCloud solution internally use the open-source solutions to execute system dynamic models in most popular model definition formats. However, we say that those solutions can use only local environment to execute models. So, we need propose approach to improve usage of local environment and add possibility to attach additional computation nodes. Also, we should make it by flexible and agile way to integrate additional open-source and internally developed tools for working with user system dynamics models and provide results to end-user by quick way.

After small review exists solutions presented in previous section of our paper we can say, that all algorithms based on following steps: read model and input data, compute each time frame and provide events to UI (if required), persist results.

Afterwards, we can define goals which can help us to build a new process of system dynamic model execution. The first goal is to split algorithm of model execution on smaller atomic parts. Reaching this goal will allow us to manipulate model execution process and build a new approach that can be applied to cloud computing cluster. The second goal is to build a new process that allow execute system dynamics models on distributed systems. Therefore, it will be possible to design architecture for cloud computing environment for executing system dynamic models – sdCloud [8], [9].

IV. Building a new process of model execution

A. Analysis of regular model execution process

The current model execution process can be described by following steps:

1) Read model structure in specific format;
2) Read input data of each model components;
3) Compute all time frames sequentially (post events about changes to UI);
4) Return modeling results.

To reach the first goal we should extract all possible atomic parts from this process.

The first part of regular execution process is reading a model structure. This part can't be split anyhow, but it can be optimized in another way. Recently, the practice of transforming the structure of a model into executable code has been used. For example, James Houghton creates solution which converts a system dynamics model definition from vensim and xmile formats to executable code on Python programming language [3], [12]. Todd Fincannon uses the same approach to execute the vensim models, this solution convert model defined in vensim format to C or JavaScript source code, compile and execute it [3]. This approach allows to optimize some execution time on reading model structure when it is required to execute the same model over and over on different sets of input data. It may seem that this optimization will bring a slight increase of performance for regular modelling process. However, if model used in pipeline of analytics on real-time data streams provided by continues modeling process, it can make a significant contribution to performance improvement [11].

The next step of execution process is reading an input data of model components. This step also can't be split to more atomics parts in general. But we can use different approaches to delivery this data from users. In this case, users can be represented by external applications that used models automatically and delivery input parameters via network channels. Some models can have small count of input parameters and users can provide those values in single batch. However, some models can have the large count of input parameters and to optimize delivery process those users can defines default values of each input parameter and then delivery to computation system only parameters that required for overriding. It can optimize communication process between user and execution system. When we are talking about optimizing a model execution process, we assume only model execution environment, and when model execution process was initialized all actual and required input parameters has already stored in data storage that should be read.

The more interested part of model execution process is computation of model time-frames sequence. There are two different levels for optimization:

1) Optimize a whole computation process of time frames;
2) Optimize a computation process of only single time frame.

When we talk about optimization of whole computation process, we assume that we take a whole process that can be computed on single node at one time. In other words, in general we can use different nodes, but at one-time moment we can use only one computation node. However, when we talk about optimization of only single time-frame computing, we can use multiple instances of computation nodes for one time-frame and one model at one-time moment. Therefore, we can use HLA terminology for this level, because this describe specifications and interfaces for distribute computing systems, which can compute one model and one time-frame on different nodes at one-time moment [13]. So, differences of those optimization levels are count of used computation nodes at one time-step. For first level of optimization it can be only one, for second level of optimization – several. And, how you can see we can't use specifications that provided by HLA for our current work. This paper is focused on first level of optimization, but our team also works in researches about next level optimization of system dynamic models [14].

The main goal of this step is computation algorithm which compute a model execution results that represented by sequence of time-frames. Time-frame is a simple data set that describes model state for a specific time in model time space. The time-frame contains values of each stock components and all required states of model components. The system dynamics model computation is based on iterative algorithms with applying the prediction and approximation functions. To compute next time-frame, we should take the results of previous computation step and apply transition operations described in model definition. Afterwards, we can repeat computations until all result time-frames for model are completely computed. So, we can say that computation of single time-frame is depends only on previous time-frame and additional metadata of model computation process. By metadata, we mean – current modeling time, modeling time range, etc. This knowledge allows us to extract this part from algorithm and makes few atomic parts. Generalizing what has been said we can split time-frames computation process on following steps:

1) For each time-frames:
   a. Gets the input data of current model execution;
   b. Gets the results of previous execution step;
   c. Compute next time-frame based on retrieved data;
2) Returns the computed time-frame results.

The last step of execution process is returning results of all time-frames. The problem of this step is that all results persist in local memory and complex models which executed during long period of time in model time space generates big volumes of data that are stored in local memory and only after execution they sent this data volume to external storages (via network or disk IO operations). And this operation requires additional time and computation resources. Also, if we will want to make some additional execution result analysis it will be possible only after completing all computation process and persisting results to external storage. The main idea for optimizing it – persist each (or almost each) time-frame separately. As mentioned earlier for computing a next time-frame required only the previous computed time-frame, respectively we can save all others computed time-frames to external storage and unload them from local memory.

B. New model execution process

Generalizing what has been said about each steps of model execution process we can show a new detailed scheme of model execution based on atomic steps:

1) Read model structure;
2) Read input data for initial state of model components;
3) For each time-frames:
   a. Gets the input data of current model execution (initial step, current time in model time);
   b. Gets the results of previous execution step;
   c. Compute single time-frame based on retrieved data;
   d. Returns the computed time-frame result;
4) Aggregate all computed time-frames;
5) Returns aggregated results.

To compare these processes, you can see the next time diagram that presented on Fig. 1.

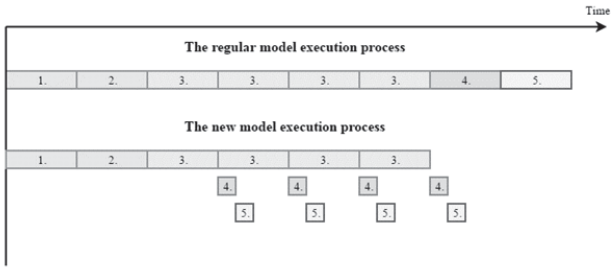

Fig. 1. Comparing time-diagram of two model execution processes (1 – reading the model structure, 2 – reading the model input data, 3 – compute single time frame, 4 – return the results of model execution, 5 – compute analytic functions)

Now we have a new process of model execution. Step with results persisting depends only on require time-frame computation instead all model execution. And we can use it for designing architecture for cloud environment.

V. APPROACH VALIDATION

To prove the described solution two system dynamic models were created. The first model describes the classic execution process of models. The second model – new execution process. We introduce two key questions to those system dynamic models:

1) When model execution process completed?
2) How many memory resources required to execute the model?

To describes processes of model execution and answer to the first question we should introduce the model time-frames flow. This flow contains 4 states:

1) Initial state – presents the state when time-frame is not computed yet;

2) Computed state – presents the state when time-frame is already computed;
3) Saved state – presents the state when time-frame was saved to external data storage;
4) Analyzed state – presents the state when time-frame through by analytical functions.

Transitions of time-frames of models are linear and described by following rates:

1) Computation rate – represents the rate of time-frames that transitioned from initial state to computed state and describes by equation (1);
2) Saving rate – represents the rate of time-frames that transitioned from computed state to saved state and describes by equation (2) and (4);
3) Analysis rate – represents the rate of time-frames that transitioned from saved stated to analyzed state and describes by equation (3) and (5).

Transition to the next state in classic execution process of system dynamic models cause only when all time-frames transitioned from previous state to the current. So, we can describe required rates, which presents above, with next equations:

$$\frac{1}{computeTime},\ InitialCnt > 0 \tag{1}$$

Where $computeTime$ – is the time in seconds of computation process for single time-frame of the model, $InitialCnt$ – is the total count of time-frames of the model that still stay in initial state.

$$\frac{networkSpeed}{fDataSize * netPacketOvr}, InitialCnt \wedge ComputedCnt > 0 \tag{2}$$

Where $networkSpeed$ – is the speed in bytes per seconds of network connection in between computation service and external data storage service, $fDataSize$ – is the size in bytes of one time frame of the model, $netPackerOvr$ – is the overhead coefficient that indicates that network transactions require use the additional information into messages, $ComputedCnt$ – the total count of time frames of the model that already computed.

$$\frac{1}{analyseTime}, InitialCnt = 0 \wedge ComputedCnt = 0 \wedge SavedCnt > 0 \tag{3}$$

Where $analyseTime$ – is the time in seconds of analyzing process that required for single time-frame of the model, $SavedCnt$ – is the total count of time-frames of the model that already saved to external data storage.

Transition to the next state in new execution process of system dynamic models performed as soon as possible. When few time-frames already transitioned to new state, but other time-frames of time model are not transitioned yet, this is not affect time-frames that already transitioned and this time-frames of the model start transitioning to the new state. And we can describe the rates with next equations:

$$\frac{networkSpeed}{fDataSize * netPacketOvr}, ComputedCnt > saveThr \vee InitialCnt = 0 \tag{4}$$

Where $saveThr$ – is the count of required saved time-frames of the model for preventing overload of the network.

$$\frac{1}{analyseTime}, SavedCnt > 0 \tag{5}$$

Those equations can help us to answer to the first question: “When model execution process completed?”. To answer on the next question, we should describe the model of the process memory. The memory it is the stock that can have two rates – income memory usage rate and outcome memory usage rate. The first rate responds to memory allocation to application. In our computation process memory required to store results of every time-frames. We can describe this rate by equation (6).

$$computationRate * fDataSize \tag{6}$$

Where $computationRate$ – is the rate of transitioned time-frames of the model from initial state to computed state, value of this rate was described above by equation (1). The outcome memory usage rate responds to free not required memory in application and this implement by garbage collector mechanism. We can describe this process by equation (7).

$$MemUsg - reqMem - gcReservation, (MemUsg - reqMem) > gcThr \tag{7}$$

Where $MemUsg$ – is the total size of memory in bytes that allocated by application from Operating System at current time, $reqMem$ – is the total size of memory in bytes that required to application correct work in fact, this value is described by equation (8), $gcReservation$ – is the size of memory in bytes that garbage collector was stay in allocation state by application, $gcThr$ – is the size of memory in bytes when garbage collector can collect and free not used memory batches.

$$ComputedCnt * fDataSize \tag{8}$$

This model of memory usage of application in computation process can answer to the second question: “How many memory resources required to execute the model?”

To run simulations, we are defined parameters of our model that presents in Table II. All models were developed in xmile formats and executed in sdCloud.io application.

TABLE II. COMPARING FEATURES FOR DIFFERENT KIND SOLUTIONS

Parameter name	Value
StepsCnt	100'000
frameDataSize	1'920 Bytes
netPacketOvr	1.04
computeTime	8 ms
analyseTime	8 ms
networkSpeed	12.5 MB per seconds
gcThr	30 MB
gcReservation	10 MB

After defining all constants that required for computing those models we can analyze results. Results of model executing presented below at Fig. 2 And Fig. 3.

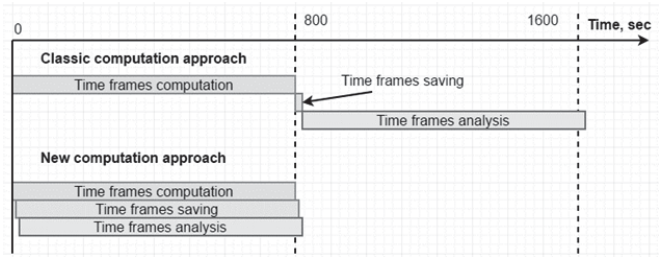

Fig. 2. Time diagram with interpretation of modelling results

Fig. 2 shows that execution process for classic approach completed near 1615 second. After it all time-frames of the model were computed, saved to external storage and all analytical functions was applied. But for new process approach execution complete near at 810 second. This value completed less than time of classic model execution process. Fig. 2 is interpretation of modelling results.

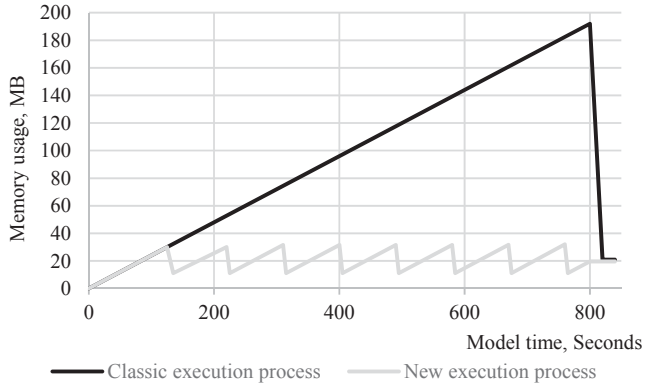

Fig. 3. Memory usage by computation application by classic and new processes

Fig. 3 shows how classic and new execution approaches used the memory of execution environment. The max value of allocated memory by classic approach requires the 192 MB. The max value of allocated memory by new approach requires the 32 MB. And this value is significantly less in comparing with classic model execution approach, where this value is six time higher.

At the current level of technology, such amount of memory is the very small piece of all memory amounts which can be used in computation node. For example, server platforms based on modern Xeon chipsets supports the 6 TB memory. However, when we talk about computations on cloud environments it is critical. Because cloud systems developed for big number of users working in parallel and this generates high load to cloud system. So, every small piece of saved resources can lead to significant overall improvement on a cloud-wide scale. Under users, we assume not only a big number of modelers who are running their models in parallel for a short period of time. Another use-case of our platform is a continuous cloud modelling when some model is running on a server for a significant amount of time and it is constantly filled with data coming from some real system. For example, IoT infrastructure of transportation system that includes sensors from cars and streets periodically generates big volumes of data. This real-data stream sent data to cloud environment to predict the state of system for monitoring and prediction of the maintenance period [11], [15], [16], [17].

Also, we should verify reliability of new computation solution. For that purpose, we should build the simple structure schemes and compute probabilities of system failure-free operation [18].

We extract four main components of computation solutions:

1) Execution service;
2) Network connection;
3) External data storage;
4) Analytical service;

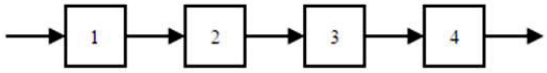

Fig. 4. Simple structure of classical computation approach

In the first case, all components of model can't be connected as parallel, we can connect only the full copy of system. By basic theory of reliability, reliability of this system that have all components in chain can be computed by equation (9).

$$p_1 * p_2 * p_3 * p_4 \quad (9)$$

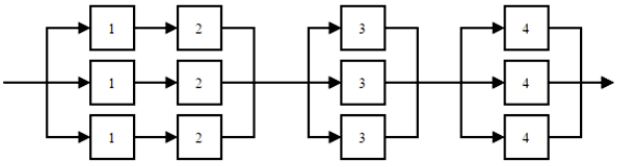

Fig. 5. Simple structure of new computation approach

In the second case, some components can be replicated by copies individually. For example, we can't provide the second network connection for computation processor by straightforward way. To solve this, we will need a specific server hardware with multiple NIC and separate network channels to destination node with data storage, and data

storage also should have the same hardware configuration. It possible, but it goes beyond the scope of this paper about software solutions. Many solutions of data storage already have possibility to run-up it in cluster with replications. In our solution we use MongoDB, but another document oriented solutions have the same functionality. So, we can use three nodes of data storage. And as described above, new process allows to continue computation process for system dynamic model from broken step, instead of rerunning it from scratch. So, by basic theory of reliability we can compute reliability of this system by equation (10).

$$(1-(1-p_1*p_2)^3)*(1-(1-p_3)^3) * (1-(1-p_4)^3) \quad (10)$$

Those equations have four parameters and for comparing these trends we should take the traits of resulting data by specific parameter. For example, we make reliabilities p_2, p_3 and p_4 are constants value is 0.9 and for p_1 we take a range from 0.15 to 0.95. Fig. 6 shows that reliability of the new computation approach higher than classic computation approach. For other trends, we have the same behavior for trends. This is proof of reliability of new computation approach.

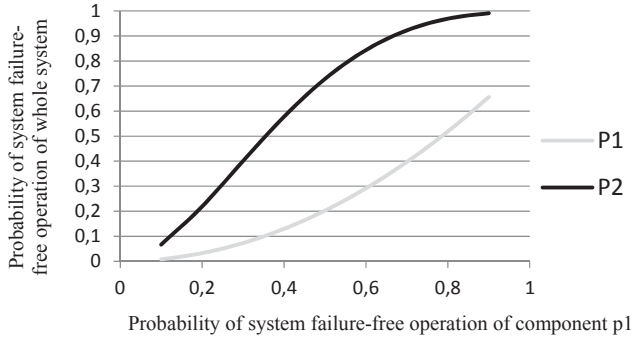

Fig. 6. Proof of computations approach reliability

VI. ARCHITECTURE OVERVIEW

As it was said earlier, to execute one time-frame of model is required to have model definition and previous time-frame results. Because of that, it is possible to execute just single time-frame and this approach also will allow us to store a time-frame results into external storage as soon as possible. If previous model execution step has been saved soon as possible to storage we can continue modeling on any other available node. This can help to solve problem of power issue or any other hardware or software faults on the current computation environment because we can move execution process to another execution environment with the least possible losses of computing resources. Another benefit of such transition of execution process is an ability to move models between nodes to optimize cloud resources usage. In modern data-centers there is an option to bring servers up and down on demand. This approach allows to significantly optimize power consumption of the deployment if the solution is hosted on real servers or monthly rental pricing for virtual servers which are working on a pricing plan like “pay-as-you-go”. For example, modelling cloud service deployment may serve requests for a time zone. In the beginning of the workday, modelling activity may grow, and cloud service will bring more and more resources up to serve everybody needs. While servers (real or virtual) will get up, models executions for everyone will be initiated on a minimal set of cloud service resources which are always up irrespectively how low current load is. Later, when new computation resources will be ready and available, already started models can be distributed to the newly available hosts. In the evening, when activities will go down, reverse process may take place. To prepare servers for shutdown, models will be transferred from low loaded servers to release them completely.

To implement this approach, we should make the independent services which can hosts on different environments and can communicates with each other via network channel. This way in software design called as micro-service cluster approach that allow to design and implement any process in a flexible, scalable and reliable way [19], [20]. These three criteria are very important in modern software development and are even more important when talking about cloud solutions. Flexibility of the architecture implemented with possibility of injection of any new additional services that can extend or improve functionality of system. Horizontally scalable systems provide an easy and reliable way of increasing their capacity based on a simple increasing of number of used executors. If one of the service in this architecture has some issues, i.e. it crashed with critical error, so performance of entire system will decrease but it is still works and take some time for operations to fix the issue and restore initial performance of the system. Also, this approach allows to use separate hosting environments and physically separate computation nodes.

To implement this solution, we should use some communication process between services, and it should be network communication because it allows to use separate nodes.

The first step of our pipeline is the service which can assign the model and its time-frames for computation to another execution service. This service takes the model and previous step of model execution and send request to start computation of time-frame of model on any available service. To make a correct decision about choosing the available execution service it communicates to each service and ask it about load and availability. To make the decision about node to compute the model uses the next parameters: model stocks count, count of required to compute time-frames, available CPU cores count and those performance. The count of model stocks and require computing time-frames influence proportionately on computation node load; and count of available CPU cores and those performance metrics influence inversely on computation node load. So, we can build decision function and select a node that have a smallest load by this function. This function presents by equation (11).

$$\frac{ModelsCount * TimeFramesCountPerModel}{CPUCount * CPUFrequency} \quad (11)$$

The next step of this pipeline is the model time frame computation service. This service takes the model structure and the previous results, after it compute the results of new time-frame and then send results to other services via distributed message queue. For preventing the network abuse all new computed time frames stored to local storage and when another decision function says to service that this local buffer is ready to send into external data storage, all this buffer moves out from this service to persist. This function should base on next parameters: size of time-frame in bytes and time of storing this data in local memory. For example, this function can look like as equation (12).

$$(LocalStoredTimeFramesCnt * TimeFrameSize > NetworkDataSizeThreshold) \vee (LocalStoredTimeFramesCnt > 0 \wedge LocalStoredTimeFrameTime > MaxSaveDelayTime) \quad (12)$$

The two configurable constants in equation can help to improve usage of network (prevent network abuse) and solve issue with not saved smallest time-frames in long computation processes. The *NetworkDataSizeThreshold* parameter allow service to send data by group in one batch that can be compressed and processed by single "call". The *MaxSaveDelayTime* allows to save small time-frames by single instances if computation process takes many time, it allows next services in pipeline to process those time-frames immediately without big blocking time.

The last step of model execution pipeline is persisting computed results into storage and computes some analytic functions. For it we introduce the separate services – Results persisting service and Analytics service. The first service takes the results of current computed time-frame, persist it to data storage.

For communication between services in pipeline we can use the special message broker based on exchanges and queues. Exchange – it's the input for any messages, can route messages between few queues by specific rules or broadcast input messages to all connected queues. The next services in pipeline connected to those queues, listen it for new messages and process when new one arrived. This approach allows us to setup any count of each service of specific type on different environments. For these purposes, we can use the next solution like as RabbitMQ or Apache Kafka [20].

Generalizing what we say about all services in model computation pipeline we can build the next diagram that presents on Fig. 7. This diagram shows each used service in architecture and communication flow based on messages.

VII. Build Proof of Concept of Described Architecture

To completely proof our solution about new model executing solution and architecture we build proof of concept which can execute and store results of modeling in specific storage. We use the following technologies to build our solution:

1) OpenSUSE;
2) C# 4.5, MONO Runtime;
3) Python;
4) MongoDB;
5) Rabbit MQ.

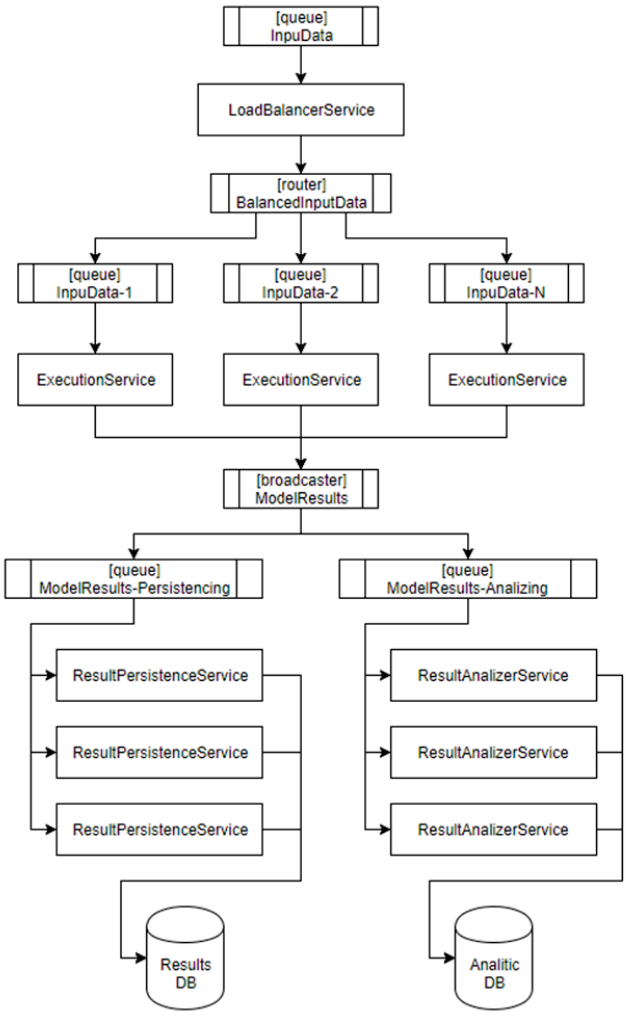

Fig. 7. Diagram of services architecture of cloud-oriented solution for systems dynamics model solution

By first iteration of our proof of concept we are implements a simple pipeline without failure simulation functions and without load-balancing. So, we build two micro-service: computation service, results persistence service. For comparing we also build the solution by classical execution process. We collect following metrics: CPU usage of process, Memory usage of process, Network usage.

By the second iteration of proof of concept we introduce low probability of system failure for each process and implement pipeline completely.

To collect all required metrics and introducing a probability of system failure we implement a simple script which can attach to specific process and collect metrics, store those data to local file with simple readable format, like comma-separated values, and check whether it is necessary to failure the attached process. For this reason, we choose the Python.

After first run of this setup we got the following results:

1. CPU usage of new solution is more biggest than classic execution process (Fig. 8), but this value is not critical and can be justified as cost for more operations that related to common functions and additional operations that should be done when solution use few services.
2. Memory usage for new solution is more stable process than classic solution, which allocate new memory ranges for each step of modelling. Memory usage are free when results are sends to results storage, but differences between max memory usage of classic execution process and new is biggest in 5 times (~55 Mbytes and ~12 Mbytes, Fig. 9). And this value depends on system dynamic model complexity. Of course, we assume that part of memory (~10 Mbytes) required for executing application itself. So, differences between memory usages can be rich to 22 times (~45 Mbytes and ~2 Mbytes).
3. Network usage for classic execution process abuse network connection and external results data storage when execution is completed (Fig. 10). While the new execution process evenly uses the network connection for persist model resulting into storage.
4. Fig. 8. show that model execution for new solution is completed near 153’000 ms, while the classic solution is completed near 154’400 ms. At this time of execution, we can sure that model execution process is successfully completed, and all modeling results is persisted into remote storage.

Generalizing all results which was got from first run of our simple proof of concept we can say that new solution required less computation resources, and those results confirm our theoretical results (Presented at section V of this paper). Memory usage was reduced in 22 times.

So, we can start to develop complete solution with all services and compare it with classical execution process on high load. We already have execution and result persistence services. To fully complete our design, we should create balancing service, configure exists services and central broker to work in parallel. Also, we should implement restore service which can able to detect broken services and try to restore it, including possibility to restore itself.

For high load testing we should generate traffic for our services with different kinds of models and input models. For this purpose, we take some models from open source libraries of system dynamic models and generates for it big bundles of input parameters. For each test run we use identical sequence of system dynamic models and input parameters for it. This turn can reduce errors in our metrics that related with the differences in input data for testing. In total for high load testing we have almost 50 models of different kind of complexity and we generate almost 10’000 correct input values for each model. In total we should execute 500’000 model runs with our testing and collect all required metrics for it.

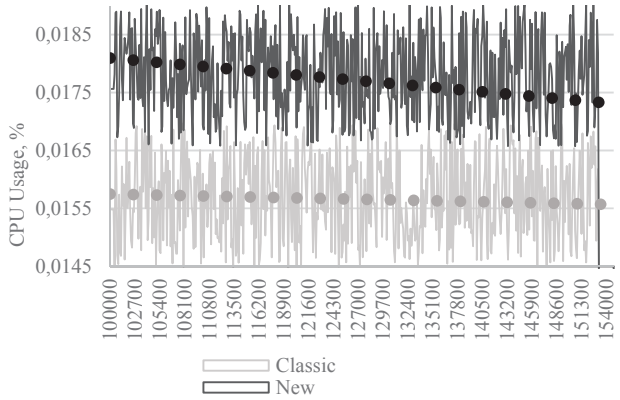

Fig. 8. CPU usage with linear approximation for classic and new model execution processes

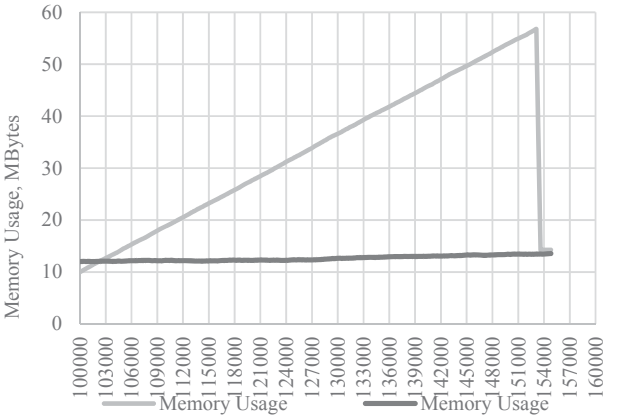

Fig. 9. Memory usage in MBytes for classic and new model execution processes.

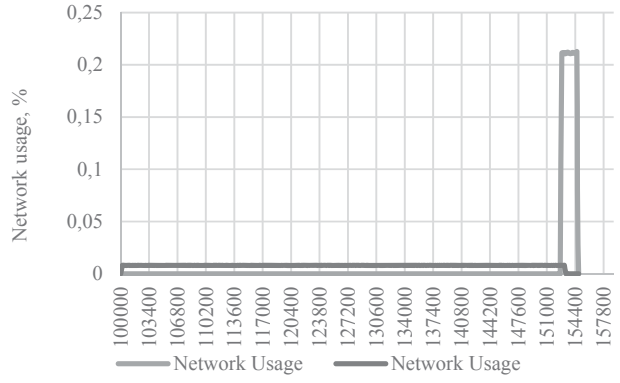

Fig. 10. Network usage for classic and new model execution processes

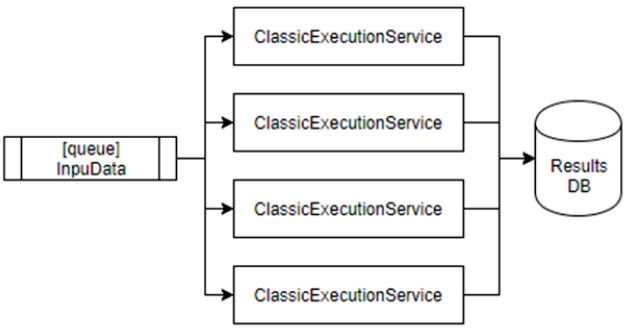

Fig. 11. Demo setup for classic execution process solution

For our purposes we build the following configuration of classic execution process: 4 execution services that attached to one queue with execution parameters which run in parallel (Fig. 11); and following configuration of new execution process: one load-balancer service that route executions to 4 execution services and one persistent service, all those services communicating via ESB (Fig. 12). All execution service for classic and new process, are attached to special Fault service and Restore service. Fault – service, it's a test service that modelling system-faults and kill specified service with specific probability value. Restore service monitor all execution services periodically (for our test solution, ping period sets to 5 seconds), and if specific service is down, restore service wake up faulted execution service again and come back to processing model execution again. As a part of this experiment we should gathered the following metrics:

1) Overall experiment time;
2) Total useful time of all execution services – CPU time that used by service for computing model results;
3) Total overhead time of all execution services – CPU time that used by service for restoring to previous state after fault;
4) Total idle time of all execution services – CPU time that used by the services in waiting for new jobs from queue.

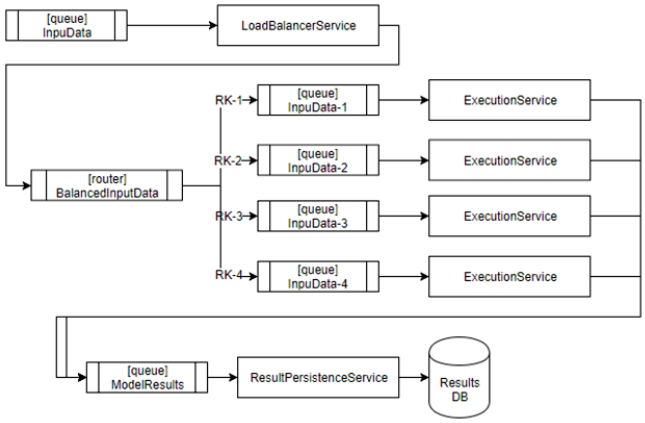

Fig. 12. Demo setup for new execution process solution

Our execution of experiments for 500'000 model executions take more than 6 hours. We run experiment on local developer machine with Intel i7 CPU (4.2 GHz, 4x2 Cores) and 16 Gb Memory. Results of experiment run is presented in Fig. 13.

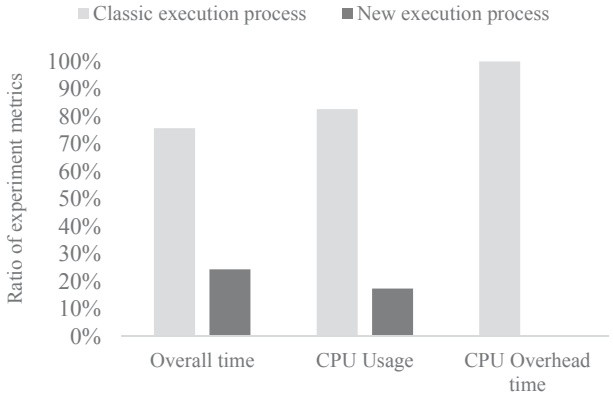

Fig. 13. Ratio of all valuable metrics from experiment run.

Fig. 13. shows that new execution process more useful for using this approach on distributed computation systems. The overall time of experiment for new execution process lower than for classic approach mostly in 3 times (4.7 hours for classic and 1.5 hours for new approaches). Wherein the usage of CPU time for execution services is reduced by 33%. The next most interesting experiment metric is a CPU overhead time, value of this metric means how many time was spent by service to restore to previous computation state after service fault. In beginning of this paper, we are described this situation and propose solution to increase reliability of execution process. Results from experiment run shows that overhead CPU time for classic execution approaches requires 25.359 seconds to restore, while the new execution process requires 0.027 seconds to restore. So, we can say that we are increase reliability of execution process of system dynamics models. When we compare these results with results that presented in section "V. Approach validation", we can see that theoretical and practical results are mostly similar.

VIII. Computing Together with Containers

Recently using the containers for hosting the applications are very popular in modern software cloud solutions. Containers isolate application in environment and contain all required dependencies and can be ran on any environment which configured to use containers. And micro-services architecture it is the first step of using the containers. When entire big and complex solution assembled from small pieces like as micro-services it very easy to use containers for each small independent service. The general profit of using the containers it possibility to dynamical scaling of each ones. Also using the container-oriented hosting such as the Amazon EC2, Google GCF or Microsoft Azure cost is cheaper than rent the dedicated or virtual servers because you can pay only to really used load.

Designed architecture of cloud oriented model computation system is fit to using it together with containers. When we discuss containers, we mean the Docker containers. Docker containers are popular, supported by community and by many hosting providers, like an Amazon [21], [22]. We can encapsulate each service into Docker containers and then we can improve our Monitor service with possibility to automatically scaling. This service should communicate with each service and monitor it load values and if required move load from one small loaded to another small loaded service and shutdown it; or in another case up the new computation service and move some load from huge loaded service to newly created service. This simple strategy just example of flexibility power of designed architecture, more detailed description how it can be implemented and really work is not fit to scope of this paper.

This approach will not only save costs on leasing hosting environments, but also improve the functionality and responsivity of the system because auto scaling allows to process many and many requests from users.

IX. Conclusion

Modeling in system dynamic is a powerful tool that has wide application range in different areas. However, existing solutions for model execution are not always simple enough for regular modelers, for example: software like a PySD or SDEverywhere. Being that execution process of system dynamic models is computation resource-consuming and vulnerable to hardware faults. We introduce a new idea how this process can be improved and used for cloud-oriented solution. Paper introduces adaptation of execution process of models, new process requires more complex infrastructure, but it allows us to provide a higher quality service for executing system dynamic models. This process more reliable in comparing with classic model execution solutions because in case of hardware or software faults the new process loose only small piece of resulting data instead losing all resulting data.

Our theoretical and practical results show that new model execution process have better metrics in comparing with classic model execution solution. For example, the new solution can better balancing network usage and reduce CPU usage by 33% in comparing with classic approach. Also, we can reduce time to restore computation to the normal state from 25 seconds to 0.027 seconds for 500’000 executions with 4 execution services and reliability of each service is 99.8%. The memory usage of new approach also is reduced in comparing with classical solution of model executions in 5 time.

References

[1] Jay W. Forester, *The Beginning of System Dynamics*. USA: MIT, 1989.

[2] Gul Zaman, Il Hyo Jung, “Stability techniques in SIR epidemic models”, *PAMM: the Proceedings in Applied Mathematics and Mechanics.*, vol. 7, 2007.

[3] GitHub website, Source code of PySD by James Houghton, Web: https://github.com/JamesPHoughton/pysd/

[4] GitHub website, Source code of SDEverywhere by Todd Fincannon, Web: https://github.com/ToddFincannon/SDEverywhere

[5] GitHub website, Source code of sd.js by Bobby Powers, Web: https://github.com/bpowers/sd.js/

[6] Vensim website, Web: https://www.vensim.com/

[7] AnyLogic: Simulation modelling for business, website: https://www.anylogic.ru

[8] sdCloud: Bringing system dynamics into cloud, website: http://sdcloud.io

[9] Perl I. A., Ward R., “sdCloud: Cloud-based computation environment for System Dynamics models”, *Proceedings of the system dynamics conference*, 2016.

[10] InsightMaker: Free simulation and modelling in your browser, website: https://insightmaker.com

[11] Perl I.A., Mulyukin A. A., Kossovich T. A., “Continuous Execution of System Dynamics Models on Input Data Stream”, *PROCEEDING OF THE 20TH CONFERENCE OF FRUCT ASSOCIATION*, 2017, pp. 371-376.

[12] J. Houghton, M. Siegel, “Advanced data analytics for system dynamics models using PySD”, *33RD INTERNATIONAL CONFERENCE OF THE SYSTEM DYNAMICS SOCIETY*, vol. 2, 2015, pp. 1436-1463.

[13] Richard M. Fujimoto, e-document “The High Level Architecture: Introduction”, Web: http://www.acm-sigsim-mskr.org/Courseware/Fujimoto/Slides/FujimotoSlides-20-HighLevelArchitectureIntro.pdf

[14] Mulyukin A. A., Kossovich T. A., Perl I. A., “Effective Execution of Systems Dynamics Models”, *PROCEEDINGS OF THE 19TH CONFERENCE OF OPEN INNOVATIONS ASSOCIATION FRUCT*, 2016, pp. 358-364.

[15] P. Marshall, “System dynamics modeling of the impact of Internet-of-Things on intelligent urban transportation”, *Regional Conference of the International Telecommunications Society (ITS)*, 2015.

[16] Abbas K. A., Bell M. G. H., “System Dynamics Applicability to Transportation Modeling”, Proceedings of the *System Dynamics Conference, Transportation Research Part A: Policy and Practice*, vol. 28, issue 5, 1994, pp. 373-390.

[17] Preventive and Predictive Maintenance, Web: https://www.lce.com/pdfs/The-PMPdM-Program-124.pdf

[18] A. M. Polovko and S. V. Gurov, *Basics of reliability theory*. SPB.: BHV-Petersburg, 2006.

[19] S. Newman, *Building Microservices: Designing Fine-Grained Systems, 1st Edition*. O’REILLY, 2015.

[20] Eberhard Wolff, *Microservices: Flexible Software Architecture*, Addison-Wesley, 2016.

[21] Rene Peinl, Florian Holzschuher, Florian Pfitzer “Docker Cluster Management for the Cloud - Survey Results and Own Solution”, *Journal of Grid Computing*, vol. 14, issue 2, June 2016. pp. 265-282.

[22] Using Docker Containers for Data Science Environments by Brittany-Marie Swanson, Web: https://www.datascience.com/blog/docker-containers-for-data-science