2015 IEEE 11th International Conference on eScience

Multi-Node Multi-Agent Cloud Simulation: Approximating Synchronisation

Antonio Giardina
Swinburne University of Technology, Australia
agiardina@swin.edu.au

Yun Yang
Swinburne University of Technology, Australia
yyang@swin.edu.au

Hai Vu
Swinburne University of Technology, Australia
hvu@swin.edu.au

Rajesh Vasa
Swinburne University of Technology, Australia
rvasa@swin.edu.au

Abstract—**Traffic engineering is a key in effective utilisation of the road network infrastructure. Simulation assists traffic engineers making informed decisions on how to operate and direct traffic within the road networks. These simulations are complex, generate big data and require high-powered computers, which can process information faster than real time, to ensure the results can be used to affect traffic. Cloud computing, a relatively new technology paradigm, can meet the essential requirements, such as scalability, interoperability, availability and high-end performance. In this paper, a novel approach to a synchronisation strategy of large-scale complex simulations is proposed. This approach builds upon advancements achieved in distributed computing. The new synchronisation strategy is designed to allow different granularities of synchronisation accuracy. Through this strategy, synchronisation overhead is reduced, thus allowing the computing bandwidth to be applied to simulation performance increases as a result of the trade off between synchronisation accuracy and performance.**

Index Terms—Synchronisation, Cloud, Computing, Traffic, Simulation, Agent Based.

I. Introduction

Traffic engineering and smart road management is a critical component of any modern city. As population grows and cities become larger, there is an increasing requirement to better manage traffic and the load it creates across motorways [1]. A key tool that assists traffic engineers to study and manage the phenomenon of traffic is agent-based simulation [2]. These simulations are complex, generate big data and require high computing power. As the size of the road network increases so does the computational complexity, computer systems are being designed to actively respond to traffic and adjust signalling to better split the load across various pathways in the network [3]. These systems use simulations to predict potential traffic hotspots and then apply traffic models to react to them. Simulations, in these cases, must run faster than real time in order for the changes to have a positive impact. As such, a concrete problem arises for the need of a computing architecture that can process large-scale simulation (LSS) faster than real time [4].

Cloud computing, a concept first introduced in the 1960's by Professor John McCarthy [5] but only recently taken place in the Information Technology world, meets the essential requirements that a simulation system for traffic engineering has. The cloud, which uses a pay-as-you-go model, is a highly scalable and highly available technology. Service providers, such as Amazon, have vast amounts of computing power that can be easily acquired to run virtual machines for virtually any computing need [6]. It is because of these benefits offered that cloud is a perfect match for running LSS. There are many tiers of cloud, IaaS (Infrastructure as a Service), PaaS (Platform as a Service) and SaaS (Software as a Service) [7]. In this paper, IaaS cloud is investigated. IaaS offers a blank virtual machine where any operating system can be installed depending on the user requirements. IaaS allows for the solution to be interoperable across many cloud providers, as it does not force the use of a specific technology interface, but rather provides the hardware required to run any type of solution. To effectively run an LSS over IaaS, an overlaying architecture must be designed to fully benefit the power of the cloud.

Various components of this type of architecture warrant further studies, such as how to handle synchronisation, how to load balance each node in the distributed system, how to divide the domain so that it can be simulated in a distributed manner [8-10]. In this paper, synchronisation of multi-node multi-agent simulation is the focus. Current approaches of synchronisation in distributed systems belong to three broad categories: centralised, conservative and optimistic [8]. These have been further discussed in Section III. Commercial cloud providers normally do not allow users to have full control over the underlying physical hardware. In most cases, the internal structure of the cloud is unknown to the user [6]. As a result cloud solutions are centred on software optimisation techniques. In an effort to deal with the black box nature of the cloud and achieve maximum performance, a novel view of synchronisation needs to be proposed. In synchronisation strategies one rule pertains to all, i.e. events in the runtime must be executed in a synchronised manner. By taking the domain into consideration, this rule can sometimes be traded off for higher performance (i.e. partial synchronisation is allowed). This is only possible in certain domains, traffic being one of them. A vital aspect of such strategy is the ability to scale up or down the granularity of the synchronisation enforcement, thus giving the ability to the user to sacrifice the level of accuracy for increased performance for each simulation run.

In summary the research proposed in this article has three distinct contributions that it intends to deliver:

- An architecture to run cloud-based simulations
- A novel method of synchronisation that enables the trade-off of simulation accuracy for performance gains in a controlled and repeatable manner
- Experiment results using existing cloud services to evaluate the architecture's performance

978-1-4673-9325-6/15 $31.00 © 2015 IEEE
DOI 10.1109/eScience.2015.32

In the following sections, Section II will introduce the motivating scenario and discuss the problem analysis, Section III will discuss the related work, Sections IV and V will propose and discuss the cloud architecture and synchronisation strategy, Section VI will present the evaluation and finally Section VII will look at the conclusion and point out future work.

II. Motivating Scenario & Problem Analysis

A. *Motivating Scenario*

Road traffic is a complex and event driven phenomenon. There are many variables that contribute to the final state of traffic in urban street infrastructure. Searching for optimised methodologies to manage traffic is an important endeavour traffic engineers undertake every day. Some of the benefits of road traffic optimisation are reduced travel times for motorists, better utilisation of road networks, reduced risk of motor vehicle accidents and vehicle emission reductions. Over the last decade many instruments have become available to the consumer that assist a driver in making the most optimal decisions while on the road. Some examples of these instruments are navigation controls with GPS, real time traffic analysis delivered through smart phones, collision alert systems, blind spot detection mirrors and speed limit indicators.

At the current rate at which science and technology are advancing, it is clear that given enough time, vehicles will become more autonomous and ultimately unmanned, e.g. Google has already developed an unmanned vehicle and is testing it live on American roads. Once unmanned vehicles become available to the consumers, it will be much easier to directly control the vehicles and safely let them reach their destinations. To effectively control each and every vehicle, a centralised system will need to exist which can identify, simulate and react to the traffic phenomena. To operate such a large-scale system, a powerful architecture will be required that can process real time data, simulate the best possible outcome and instruct vehicles to choose the correct paths to reach their destinations. This presents us with the perfect test-bed for running a simulation in the cloud. The simulation will be complex, require scalability and need to process information and return results efficiently and in a timely manner.

B. *Problem Analysis*

As introduced in Section I, the principal outcome for the research carried out in this paper is to develop a software architecture that is able to run LSS at the highest performance level possible. As discussed in Section II.A, a possible use of such architecture is the operations of a road network completely used by unmanned vehicles. The cloud-computing paradigm will be adopted for the underlying hardware technology. Architecturally, the cloud behaves much in the same way as distributed architectures such as grid or cluster computing. As such, many principles that apply in distributed computing can also be applied to the cloud. Two major distinguishing differences of cloud computing are that hardware is shared amongst many users and it functions like a black box as users do not have full control of the underlying setup. Due to this shared black box nature, a distinct problem arises when constructing cloud software architecture, i.e. the software must be able to detect initial signs of the underlying hardware, such as increase in the overall resource usage, and react accordingly to adjust the operations of the application.

There are various software components that require analysis when constructing architecture to run LSS in the cloud, as discussed in Section I. In this paper, the synchronization strategy is the primary problem of focus. To operate this distributed system, synchronisation of various nodes present within the system is a primary objective. Without synchronisation, complex software applications such as a simulation, cannot function. For example, in a road traffic simulation, the geographical area where vehicles can move freely is split into different parts and each part is placed in a node. Without any synchronisation mechanism, vehicles moving from one node to another would be able to move forward or backward through time uncontrolled, i.e. if one node is ahead in the simulation compared to other nodes, it is also essentially ahead through time compared to these same nodes. This uncontrolled synchronisation would also not be detectable and would invalidate the simulation. The synchronisation process can be regarded as an overhead to the primary goal of the simulation, which is to simulate a road traffic network. If synchronisation is viewed as an overhead, it becomes evident that by reducing the amount of computing resources the synchronisation method requires more resources will be available to improve the simulation performance. As such in this paper there will be a strong focus on reducing this synchronisation overhead that will be achieved by trading accuracy for performance.

III. Related Work

Over the last few decades simulation and computers have shared a close bond [11]. As simulations have become larger and more complex, the need to use high-powered computing (HPC) has increased dramatically [12]. Currently the principle HPC hardware that is used to run large-scale simulation (LSS) is super computers [13]. These custom built machines are both expensive and limited in availability [14]. Due to these limitations there has been an effort to find other computer hardware solutions to run LSS [15]. Distributed system is one of these venues. Agent based simulations (ABS) is one type of LSS that may require HPC, and the one that will be explored in this paper. Important to this research are the common issues faced when trying to run ABS in a distributed environment. Some of these issues are: how synchronisation is achieved, how the model is load balanced over the distributed architecture and how the model is subdivided and split across the distributed system.

With the advent of cloud computing, a new distributed system paradigm, research has been done to exploit the advantages cloud brings such as scalability, interoperability, availability, performance and low cost to run simulations. In [16], the authors attempt to combine an evolutionary agent model with the cloud infrastructure. Using the MapReduce programming model, iterations of the evolution model are divided across different nodes. This approach becomes less effective when the simulation grows in size. As each node cannot run the entire iteration of the simulation in a time effective manner, the iteration itself must be distributed into multiple nodes and as such a new approach must be found.

Authors in [17] introduce an adaptation of High Level Architecture (HLA) to run LSS in the cloud. HLA is a standard for running simulations on distributed systems. As HLA is not directly suited for the cloud, the authors investigate new means to deal with load balancing and effective use of cloud resources. The research carried out by the authors of [18] presents us with the closest set of goals related to this research. In this article the authors investigate how a social agent model can be distributed over the cloud. They look at the division of the model environment over multiple nodes, how to synchronise the model and process data on adjacent nodes. They only present a feasibility study of the cloud with some preliminary results.

Throughout the papers thus far discussed three major areas of research arise that are independent of the approach but are caused by the distributed nature of cloud: synchronisation [8], load balancing [10] and domain model division [9]. In this paper the main area of study is synchronisation. The intent of the research is to offer an optimised strategy to handle synchronisation and reduce overhead brought in by applying strategies to the architecture. Synchronisation of distributed systems has been a well-studied area [8]. By viewing discrete event based simulations [19] at a higher abstraction level, advances achieved in this field can be directly applied to agent-based simulations. Each computer node in the distributed architecture can be seen as a logical process (LP) with the agents that move from one node to another as the events. This view of the system allows us to abstract away the internal workings of a node and look at the system as a whole.

In discrete event based synchronisation [8] there are three main families of synchronisation: centralised, conservative and optimistic. In centralised synchronisation approaches [20], a central control mechanism exists which serves as the master clock for maintaining all LP synchronised. In conservative synchronisation approaches [21], all LP synchronise with one another, no LP can fall out of synchronisation with others and the process of synchronisation cannot create any deadlocks. Within conservative synchronisation approaches there a many variations of the strategy that are optimised for either the domain or architecture. In optimistic synchronisation [22], LPs are allowed to progress with the execution of events. The overall system must still remain synchronised, but contrary to conservative methods, the system is synchronised in a reactive manner. If one or more LPs are found to be out of synchronisation, all events that occurred are rolled back to the latest most synchronised state.

In section VI.B.2) and VI.B.3), the centralised and conservative methods of synchronisation have been further explained. These strategies have been applied to the cloud architecture created to evaluate the findings of this paper and will serve as the benchmark measurements of performance for comparison. Optimistic synchronisation methodologies have not been selected as viable synchronisation strategies due to the nature of the underlying simulation. As optimistic strategies rely on rollbacks to resynchronise all LPs and in the case of this research one LP is an agent-based simulation, the number of rollbacks required for resynchronisation would result in a net performance loss.

A concept that has proven to be vital in the development of the strategy proposed in this research is the concept of time windows [23]. In time window synchronisation of two LPs can become unsynchronised by a maximum predefined amount. Events can occur while the two LPs are unsynchronised and will not be rolled back if they do occur. It is though important to understand how this affects the simulation macro results and have the ability to track the amount each LP has been influenced by unsynchronised events. Built upon the concept of time windows and discussed in detail in Section V is the new concept of simulation lag. This becomes the basis of this paper on how accuracy can be traded off for performance increases of the simulation.

IV. ARCHITECTURE FOR SIMULATION AS A SERVICE

Simulation as a Service (SIMaaS) is the idea of providing simulation as a new provisioning of the cloud, turning simulation as a pay-as-you-go on-demand service available to all. In order to provide SIMaaS, a supporting architecture must be developed that is implemented on the cloud. As such, Sim Net Kay (SNK) has been constructed. The software architecture of SNK is both modular and distributed. It is modular because components within the architecture automatically activate when required. It is distributed as it can be deployed across multiple computer nodes, i.e. each virtual machine acquired from an IaaS cloud.

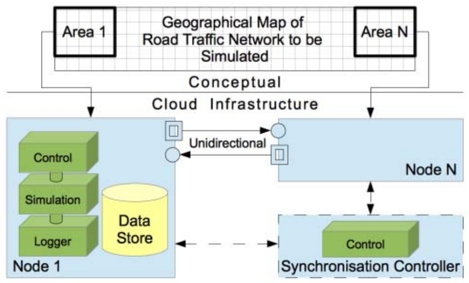

Figure 1 The overall distributed architecture of SIMaaS

Figure 1 describes the high level view of the SNK architecture. It demonstrates how the domain layer is subdivided and processed by individual nodes within the underlying architecture, i.e. how the geographical area of a traffic simulation is split and mapped to each individual node in the underlying infrastructure.

SNK requires a mapping between the simulation domain (the area of road network to be simulated) and the underlying infrastructure (each individual simulation node). To effectively run a distributed simulation the first step is to select a method for splitting the domain into smaller distributed chunks. In the case of road traffic simulations, the geography can be used to segment the simulation.

Area 1 will envelop all roads, intersections and vehicles present within it and be responsible for simulating any actions that occur within the boundaries. An important property about vehicles, compared to roads, intersections and motorways, is the fact that they do not belong to one area for the entire simulation. Vehicles might remain within the area but can also move freely from one area to another as they progress towards their desired destinations. The domain is split before the simulation begins and each node will load the specific area when the simulation is initialised.

Each processing node in SNK has a baseline structure. This structure remains the same across all nodes within the cloud system. As shown in Figure 1 a node has four main components: control, simulation, logger and data store. Connecting the node to external components of the overall system are unidirectional sockets. The control component is charged with enabling the node functionality. It controls functions such as initialising the node, creating the socket connections to other nodes, initialising and starting the simulation. As the name describes it controls the infrastructure surrounding the simulation. The simulation component allows the simulation to be run over the cloud. This component has been created with the purpose of being a plug and play bucket. Any traffic simulation tool that can be distributed across multiple nodes could be integrated into SNK via this component. It allows the architecture to be abstracted away from one single simulation type. The logger is charged with logging events and handling all I/O operations. The logger can lag behind the simulation as I/O events can slow down processing times. The unidirectional sockets use a TCP protocol for the method of communication to ensure no packets between nodes are lost. The synchronisation controller is a component of the synchronisation strategy and will be further discussed in Section V. All components have been created separately from one another, as they have been designed to run on their own CPU thread. This has been done to minimize the impact they might have on each other.

V. Strategy for Synchronisation by Approximation

There are various methods to achieve synchronisation of a system. As discussed in Section III, these methods can belong to centralised, conservative or optimistic methodologies. This section introduces a hybrid strategy that employs ideas from both centralised and conservative methodologies. The primary goal of this strategy is to reduce the overhead that is caused by enforcing synchronisation in a system. This reduction will result in a net performance increase of the overall software. Centralised and distributed strategies will serve as comparison benchmarks and are discussed in Section VI.B.

The synchronisation strategy proposed in this paper trades off synchronisation accuracy for performance increase. The trade off varies and is controlled by the user running the simulation. This trade off is possible because of a key principle in the domain of traffic simulation. In road traffic simulations the phenomenon of traffic or traffic jams is a result that occurs at the macro level. As such it is the net sum of many vehicles contributing to the escalating traffic, a single vehicle has a minimal effect on the overall result. The primary outcome for synchronising two adjacent simulation nodes is to ensure that vehicles, moving from one node to the other, do so without travelling through time (i.e. *node 1* is at time interval 10 and *node 2* is at time interval 12. If a vehicle moves from *n1* to *n2* it would effectively travel into the future). If we relax the constraint that vehicles cannot travel through time, this allows the system to reduce the overhead caused by maintaining the simulation synchronised every tick of the runtime.

From a simulation perspective if a vehicle travels through time by a small factor compared to the overall duration of the simulation (i.e. thirty seconds compared to one hour) the impact to the macro phenomena will be negligible. With the reduction of the synchronisation overhead more resource can be assigned to increasing simulation performance and thus this will allow for faster real-time simulations. As discussed in Section II if the intention of the simulation is to affect traffic in an attempt to reduce it, having quicker but less accurate results, will facilitate the initial identification of traffic hotspots. This will allow for a quicker response to the phenomena, which then can be further controlled with more accurate longer running simulations that would have a higher or complete degree of accuracy.

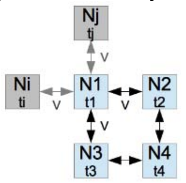

Figure 2 Approximation synchronisation strategy node topology

There are two main components that drive the synchronisation strategy: proactive and reactive controls. These two control mechanisms combined together give users control over the desired accuracy of the simulation. Figure 2 outlines the topology of nodes within the cloud architecture. Each node is assigned a geographical area and vehicles (V) can move freely from node to node. Once all nodes are initialised, the simulation begins on each node and runs freely until either proactive or reactive controls activate.

The first control component is a proactive control. This is because it controls the synchronisation process actively as the simulation is run. It is based on the principle of time window where two different processes are allowed to fall out of synchronisation up to a certain maximum window.

$$TW = |t_D - t_A|$$

The equation above describes the time window (TW) as the absolute difference between the departure (t_D) and arrival (t_A) times of an agent (vehicle) moving from one node to another (i.e. at what simulation time tick did the agent leave the first node, and at what simulation time tick did the agent arrive at the second node).

$$TW \leq TW_{MX}$$

There exists a maximum time window allowed (TW_{MX}). If agent movement occurs with a time window greater than the maximum, the agent and the arrival node are halted, till the time window falls back into a range of less than the maximum. An important architectural feature is that no additional layer of communication is added for synchronisation. All synchronisation events are piggy backed on the agent moving from one node to the other (i.e. vehicle objects carry synchronisation data within them). As the simulation is allowed to fall out of sync, agent movement offers enough reoccurrence to enforce the synchronisation strategy.

The second control component is reactive. This is because it is based on the result of an algorithm calculated after each tick of the simulation. The algorithm returns the synchronization lag coefficient (SL). SL is a value between 0 and 1, where 0 indicates a fully synchronised simulation. In the following paragraphs the calculation method for SL will be explained. All equations are calculated and updated each tick of the simulation. One tick symbolises one unit of simulation progress calculation. All equation results are node specific and are calculated locally in each node.

$$A = A_{Initial} + Y - Z \qquad A_C = A_{Initial} + Y$$

$A_{Initial}$ denotes the initial number of agents (vehicles) in a node before the simulation begins. A denotes the number of agents currently in a node. Y is the number of agents that have entered the node and Z is the number of agents that have exited the node. A_C is the cumulative amount of agents that have existed within the node for the entirety of the simulation.

$$dTT = \frac{TW}{TW_{MX}}$$

The degree of time travel (dTT) is calculated by dividing the time window of the agent, which has travelled through time, by the maximum time window allowed. This value provides the first set of information in identifying how unsynchronised the simulation is.

$$Os_T = \sum (1)\ where\ t_D \neq t_A\ for\ a \qquad Os = \sum Os_T$$

Os_T denotes the count of agents (a) that travelled where the departure time (t_D) did not equal the arrival time (t_A) (out of sync). Os is the overall count for the entire simulation.

$$DTT_T = \sum (dTT)\ where\ t_D \neq t_A\ for\ a \qquad DTT = \sum DTT_T$$

DTT_T denotes the overall degree of time travel for all agents (a) in the node that travelled through time (out of sync). DTT is the overall degree of time travel for the entire simulation for the node.

```
A - Simulation Node Initialisation
1   Load configuration settings
2   Establish connection with synchronisation controller
3   If local TCP server has been started
4       Send ready alert to synchronisation controller
5   If go command is received from synchronisation controller
6       Create and connect TCP clients to each adjacent node
7       Begin Simulation

B – Synchronisation Controller Initialisation
1   Load configuration settings
2   Initialise local TCP server
3   If ready alert has been received from all Simulation Nodes
4       Send go command to all Simulation Nodes

C - Simulation Node Run Operation (First Tick)
1   Load assigned traffic map into simulation
2   Load agents into simulation
3   Carry out simulation calculations for the current tick
4   Send all agents that are moving from local to adjacent maps to
    the correct node

D1 - Simulation Node Run Operation (Subsequent Ticks) –
Concurrent Process 1 (CP1)
1   Insert agents received (CP2) into local map
2   Calculate current tick of the simulation
3   Send all agents that are moving from local to adjacent maps to
    the correct node

D2 - Simulation Node Run Operation (Subsequent Ticks) –
Concurrent Process 2 (CP2)
1   Receive agents as they are sent from adjacent nodes
2   If the TIME WINDOW is bigger than allowed
3       Halt simulation (CP1)
4   Else if the SIMULATION LAG is bigger than allowed
5       Perform re-synchronisation step
6   Else
7       If simulation is halted
8           Begin simulation (CP1)

E - Simulation Node Total Re-Synchronisation
1   If re-synchronisation command is activated locally
2       Halt simulation (CP1)
3       Send alert to synch controller with current tick count of sim
4       Send alert to all adjacent nodes
5   If re-synchronisation command is received from synch
    controller
6       Halt simulation (CP1)
7       Send current tick count of simulation to synch controller
8   If re-synchronisation max tick received from synch controller
9       Begin simulation (CP1)
10      Simulate till max tick and then halt simulation (CP1)
11      Send alert to synchronisation controller
12  If re-synchronisation complete received from synch controller
13      Clear all SL calculation values and begin simulation (CP1)

F – Synchronisation Controller Total Re-Synchronisation
1   If re-synchronisation command received
2       Calculate max tick from all nodes and send to all nodes
3   If max tick reached from all nodes
4       Send re-synchronisation command to all nodes
```

Figure 3 Pseudo code of synchronisation strategy

$$ATT = \frac{Os}{A_C} \qquad ATW = \frac{DTT}{Os}$$

ATT denotes the overall percentage of agents for the node that have travelled through time. ATW denotes the degree of time travel these nodes have done based on a percentage of the maximum time window, i.e. how much they have been allowed to travel out of sync compared to the maximum out of sync allowed.

Given the above equations it is possible to calculate the synchronisation lag coefficient SL.

$$SL = ATT \times ATW$$

Figure 3 illustrates the pseudo code of the strategy described and how it is applied to the cloud architecture.

Figure 3 Parts A and B illustrate the steps required to initialise the overall architecture on the cloud. As each node must connect to all other adjacent nodes and the synchronisation controller, it is vital that all sockets are initialised properly, i.e. server component of the socket started before the client.

Figure 3 Part C illustrates the first step required in starting the simulation. This step will load all data and perform the first tick of the simulation. Processes outlined in Parts D1 and D2 happen concurrently. D1 illustrates the execution of the simulation tool as it processes the simulation. D2 controls the synchronisation process, this is done by receiving the agent, verifying based on the strategy algorithm the current synchronisation state and delivering the agent to be processed in the simulation.

Figure 3 Parts E and F illustrate the reactive mechanism of the synchronisation strategy. If the simulation reaches an SL that is greater than the desired one, the simulation is resynchronised.

In Figure 3 the hybrid structure of the strategy can be identified. It is conservative as nodes synchronise with each other via the agents (vehicles) moving through them. It is also centralised, as the central synchronisation controller handles initialisation and re-synchronisation.

At the completion of the strategy described in Figure 3, all simulation results are logged, and the simulation is allowed to complete closing down the entire process on the cloud. Simulation results can be either stored locally or on cloud storage infrastructure. They can also feed into a traffic management computer system, to influence the current state of traffic in an attempt to alter the formation of any traffic hotspots.

VI. Evaluation

In order to measure the performance gains achieved by the strategy proposed in this paper, two vital sets of experiments were conducted. The first experiment was to choose an appropriate cloud node hardware setup that would not affect the outcome of the experiment due to any instability caused by factors other than our synchronisation strategy. The second experiment was to measure the performance gains by benchmarking the strategy against centralised and conservative methods. In this section these experiments will be discussed in detail.

A. *Measures and Attributes*

In other to evaluate performance gains of the synchronisation strategy the unit of time taken has been selected. Time taken (i.e. the end clock time minus the start) will be recorded and compared throughout each experiment. The decision to use time taken as the unit of measure is based on the following factors:

- *The traffic simulation is constant: Each agent in the simulation has been programmed to take the same path for each run of the simulation; each road and traffic light have the same attributes governing its behaviour (i.e. road speed limit, traffic light cycle time).*
- *The simulation attributes are constant: The time to be simulated is the same for each simulation run (i.e. the number of simulation ticks to perform); the number of agents present in each node at the start of the simulation is the same.*

Given these repeatable behaviours and attributes, each simulation run, should equate to the same net result and will only be affected by architecture, strategy or hardware factors. Furthermore, to ensure accuracy, each experiment is run multiple times and the resulting distribution is analysed.

B. *Benchmark*

In order to measure the overhead reduction and performance increase of the strategy proposed in this paper, a benchmark must be established. There are two important parts of the architecture that require benchmarking. The first is the cloud infrastructure itself. As there are many types of hardware setup for cloud nodes available selecting the appropriate one is a priority. The second is to benchmark well-known representative synchronisation strategies. These strategies will follow closely centralised and conservative approaches and will serve as a measure to compare results.

1) *Cloud Infrastructure*

Amazon AWS has various types of cloud infrastructure that can be chosen and used, which belong to different families: micro instances, general purpose, compute optimised, memory optimized, GPU and storage optimized. For the purpose of this paper, Amazon AWS micro, general purpose and compute optimised were selected and tested. The primary goal of the benchmarking was to select an instance which was least affected by variations in its performance. A single desktop computer was also tested to benchmark the appropriate variance on a single machine.

TABLE 1 PERFORMANCE IN SEC. OF VARIOUS CLOUD INSTANCE TYPES

Type	Low	Q1	Median	Q3	High
Desktop	2121	2171	2220	2272	2337
Micro	986	2594	7205	11481.5	14440
General Purpose	2550	3274.5	4000	4478	5695
Comp Optimized	384	396	409.5	421.5	458

As depicted in Table 1, a single machine entirely dedicated to running the simulation has a performance variation of 9.2% of the total time taken to run the simulation. Micro and general-purpose instance types have a variation of 93.1% and 55% respectively. These variations are too high and would leave any experimentation with uncertainty on the results. The compute optimised has a variation of only 16.1%. It is the lowest variation and the closest to running a single machine. This type of CPU intensive instance has been selected for experiments.

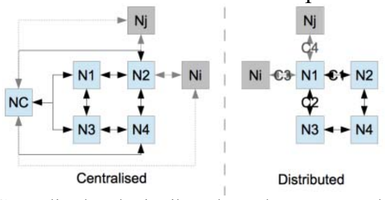

Figure 4 Centralised and Distributed synch strategy node topology

2) *Centralised Synchronisation Strategy*

In a centralised approach to synchronisation, all parts of the system are kept in sync by a centralised control structure. At any point of time, the system is fully synchronised and no events can occur out of sync.

In Figure 4 (left), the node control (*NC*) is the central node that controls the overall synchronisation of the system. For each tick of the simulation a message is sent from *NC* to each node, i.e. Node 1 (*N1*), to begin the processing of the simulation tick. Once *N1* has finished processing the tick a message is sent back to *NC*. Once all nodes in the system have replied to *NC*, *NC* will issue the command to begin the next tick in the simulation. Each node in the system is connected to both *NC* and other adjacent nodes. The connection amongst nodes is only used for the communication of movement of a vehicle from one geographical area to the next.

$$SC_t = 2N$$

The above equation describes the overhead present at *NC*. The number of synchronization computations per tick (SC_t) of the simulation is twice the number of nodes (*N*) present in the system. This is due to a node, *Nj*, requiring a computation to begin the simulation tick and another computation when that simulation tick is completed. When the simulation is small enough, this is not a major issue, but as the simulation becomes larger and the number of nodes required to process the simulation increase so does this overhead. This overhead is a problem because it is localised at one single point, *NC*, and as such will impact the overall simulation performance.

3) *Distributed Synchronisation Strategy*

The distributed synchronisation strategy discussed in this section has been adopted from the conservative method of synchronisation. In this type of methodology the system being synchronised must follow two important rules, no event can occur out of sync of one another and any dead lock that may occur must be prevented.

Figure 4 (right) outlines how the distributed synchronisation is achieved. All nodes synchronise with their adjacent nodes. A node cannot progress to the next tick of the simulation till all nodes adjacent to it have completed the current tick (i.e. *N1* must wait for confirmation from *N2*, *N3*, *Nj* and *Ni* before it can proceed to the next tick). The system of nodes can be as large as required, and this type of strategy will continue functioning. The synchronisation will spread throughout the network of nodes and the simulation will progress.

$$SC_t = 2C \text{ where } 1 \leq C \leq 4$$

The above equation describes the overhead present at each node in the simulation system. The number of synchronisation computations per tick (SC_t) of the simulation is two times the number of adjacent nodes (*C*) that a node has. As discussed in Section IV, the domain has been split by geographical boundaries, the maximum number of adjacent nodes a node can have is four as the system is a two dimensional grid. The number of computations required is based on the principle that each node must both inform and be informed when the current tick of the simulation has been completed. The distributed synchronisation method greatly improves the synchronisation overhead required as it distributes this computational load across the network of nodes and decentralises it. This though is more evident when the number of nodes used is very larger and the difference in synchronisation computation time is noticeable.

C. *Simulation Results*

To evaluate the performance improvements achieved by the synchronisation strategy proposed in this paper, three types of experiments were run. The first experiment analysed the effects of increasing the number of distributed nodes within the system (Section VI.C.1)). The second experiment investigated the effect of the number of agents present in each node at initialisation (Section VI.C.2)). The third and final experiment was run to analyse the efficiency of synchronisation by approximation at various time windows and simulation lags (Section VI.C.3)). Throughout the experiments the principal theme was to trade accuracy for performance. This was achieved by allowing neighbouring nodes to fall out of synchronisation and still allow vehicles to move between them. As explained in Section V, the granularity of accuracy was controlled by the value of time window and simulation lag coefficient. For the first two experiments (Sections VI.C.1) and VI.C.2)), the time window was set to 60 seconds and the simulation lag coefficient to 1 (further explanation on the effects of these two values can be found in Section VI.C.3)).

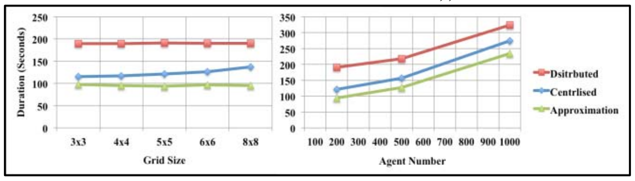

Figure 5 Graph plotting the average performance for each synchronisation strategy at: (A) each grid size and (B) each agent number

1) *Simulation of Grid Size's Effect on Performance*

The simulation grid size is the number of nodes used to achieve the simulation. As discussed in Section IV the overall traffic network map is split into geographical chunks and assigned to each node for simulation. Depending on the size of the simulation more and more nodes might be required to achieve the simulation within acceptable real-time limits. For the purpose of this experiment five grid size types were selected: 3x3, 4x4, 5x5, 6x6 and 8x8. These grid sizes were selected as they offered both diversity and a realistic size variance, from 9 to 64 nodes, the latter being able to simulate vast traffic network regions.

A key point to remember is that even though the grid size is increasing the load on each individual node, brought in by the simulation itself, is close to constant as the nodes are initialised with the same size geographic map and same amount of agents. Therefor the difference in performance is caused by the synchronisation strategy overhead.

As discussed in Sections VI.B.2) and VI.B.3) and demonstrated in Figure 5A both the centralised and distributed synchronisation strategies are behaving as predicted. In the centralised approach the performance overhead grows in correlation with the increase in number or nodes. In the distributed approach the synchronisation overhead will only grow if the number of connections to other nodes grows. As such from 3x3 to 8x8 the number of connections from one node to its adjacent is constant and therefore the resulting overhead is constant too.

Furthermore, as shown in Figure 5A, synchronisation by approximation out performs centralised and distributed strategies at every grid size. By relaxing the synchronisation requirements (i.e. accuracy vs. performance) the overhead can be vastly improved and as the grid becomes bigger this improvement also increases when compared to the centralised approach. The resulting simulation, albeit less accurate as vehicles were allowed to travel in time, still demonstrated the traffic phenomena required to make informed decisions on the state of the traffic network.

2) *Agent Size's Effect on Performance*

The simulation agent size, in the context of this experiment, is the number of agents representing vehicles in each node at initialisation of the simulation. A 5x5 grid structure has been selected to analyse the effects of agent size. In Figure 5B it is clearly visible that agent size has little impact on the various strategies themselves. Even though there is a performance reduction, i.e. the simulation is taking longer to complete, the increased number of computations required to simulate all extra agents causes it. Therefore it can be stated that the increasing number of vehicles in a node has no impact on the synchronisation strategy overhead but rather just an impact on the simulation performance itself, i.e. time taken for a sim to compute.

3) *Synchronisation Accuracy's Effect on Performance*

There are two main values that can be set to control the synchronization by approximation strategy. As discussed in Section V, these are the maximum allowable time window and the simulation lag coefficient. Depending on the level of accuracy required these can be adjusted to increase or decrease the granularity of the resulting simulation accuracy. The maximum allowable time window controls the amount of non-synchronisation that can exist between two nodes, i.e. how far ahead in the simulation can one node be compared to the other? The simulation lag coefficient takes a wider look at what has already occurred throughout the simulation and then controls when the simulation must re-synchronise due to it exceeding its allowable maximum. A value of 1 in SL indicates that all agents that have existed in a node have firstly travelled through time and secondly have travelled with a time jump equal to the maximum allowable time window. For SL to equal a value of 1, the node would need to start with no agents and all agents entering the node would need to be traveling at the maximum allowable time window. Based on this, if an empty node is present in the simulation, agents would be more likely to travel at the maximum time window as the value of the time window was made smaller.

For the purpose of this experiment, grid and agent sizes respectively were set to 5x5 and 200. For the time window - simulation lag values, the following were tested: 60 - 1, 30 - 0.5 and 15 - 0.25. As the values become smaller, the level of accuracy in synchronization increases, i.e. vehicles can move less through time and less overall vehicles are allowed to move through time.

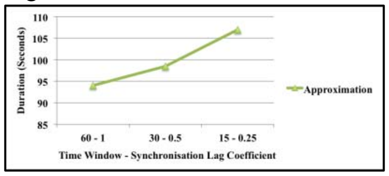

Figure 6 Performance results for various control values applied to synchronisation by approximation

Figure 6 demonstrates that, as the level of accuracy of the synchronisation increases, the overall performance of the simulation decreases. This is an expected result as our performance improvements derive from the trade off with the level of accuracy of the synchronisation strategy. Recalling the results in Figure 5A, the median results for the duration of the simulation (with the same grid and agent size) for centralised and distributed strategies respectively are 121 and 191 seconds. Even at higher levels of accuracy, synchronisation by approximation outperforms centralised and distributed strategies.

D. *Discussion*

Demonstrated by the simulation results discussed in Section VI.C, synchronisation by approximation outperforms both centralised and distributed strategies in terms of performance. By allowing for different levels of synchronisation accuracy, simulations can run faster by trading off accuracy for performance. In the modern day scenario this can strongly help achieve requirements, which otherwise, would be very hard and costly to overcome, e.g. affect traffic in real time.

Even though the immediate results of the simulation may not be always 100% accurate, they can provide enough information to identify early symptoms of traffic congestion formation. Changes can then be applied immediately to hotspot areas, while a longer-running more-accurate simulation confirms both the prediction and outcome of these changes. The variable accuracy level is also highly important as many simulations can be started concurrently but will finish at different time intervals, all providing a more accurate result of the previous simulation.

VII. Conclusion and Future Work

The area of traffic management is vital to the running of all major cities. Computer assisted simulations enable traffic engineers to identify the creation of traffic hotspots. These simulations must run faster than real time if the results are to be used to affect the state of traffic. Cloud computing is a technology paradigm that can be adopted to run such simulations. It has been demonstrated in this paper that by applying a new method for controlling synchronisation of a simulation running in the cloud, performance can be improved by reducing the synchronisation overhead and diverting this overhead to performance gains, a vital step in assuring faster than real time capabilities. This is achieved by the novel architecture proposed by this research, which enables the trade off of simulation accuracy for performance gains (feasible due to the domain of traffic) in a controlled and repeatable manner.

The synchronisation strategy proposed in this paper can be further improved by looking at factors such as load balancing and domain model division. If the simulation load distribution can be further optimised both proactively (domain model division) and reactively (load balancing) further performance gains can be achieved. Furthermore, constructing a multi-layered architecture for varying accuracy can assist the simulation framework proposed in returning increasingly accurate results over time, thus allowing better control of the traffic phenomenon.

Acknowledgment

The research team thanks Amazon (AWS) for the support provided with the Education Grant Award.

Reference

[1] P. Croft, Guide to Traffic Management, Austroads, 2009.

[2] T. Cheng & H. Shengguo, 'An Extensible Multi-agent Based Traffic Simulation System', Int. Conf. on Measuring Technology and Mechatronics Automation, pp. 713-716, 2009.

[3] H. Kirschfink, J. Hernández & M. Boero, 'Intelligent Traffic Management Models', ESIT, pp. 14-15, 2000.

[4] Y. Nakai, D. Perrin, H. Ohsaki & R. Walshe, 'Performance Evaluation of Cloud-Based Parallel Computing', IEEE Annual Computer Software and Applications Conf. (COMPSACW), pp. 351-355, 2013.

[5] J.F. Ransome & J.W. Rittinghouse, 'Front Matter', in Cloud Computing, CRC Press, 2009.

[6] M.A. Vouk, 'Cloud computing - Issues, research and implementations', Int. Conf. on Information Technology Interfaces, pp. 31-40, 2008.

[7] J.F. Ransome & J.W. Rittinghouse, 'Web Services Delivered from the Cloud', in Cloud Computing, CRC Press, 2009.

[8] S. Jafer, Q. Liu & G. Wainer, 'Synchronization methods in parallel and distributed discrete-event simulation', Simulation Modelling Practice and Theory, vol. 30, pp. 54-73, 2013.

[9] M. Scheutz & P. Schermerhorn, 'Adaptive Algorithms for the Dynamic Distribution and Parallel Execution of Agent-Based Models', Journal of Parallel and Distributed Computing, vol. 66, pp. 1037-1051, 2006.

[10] M. Randles, D. Lamb & A. Taleb-Bendiab, 'A Comparative Study into Distributed Load Balancing Algorithms for Cloud Computing', IEEE Int. Conf. on Advanced Information Networking and Applications, pp. 551-556, 2010.

[11] R.P. Wolfson & J.H. Gower, 'The Role of Computer Modeling and Simulation in Electric and Hybrid Vehicle Research and Development', IEEE Trans. on Vehicular Technology, vol. 32, pp. 62-73, 1983.

[12] J.A. Doornik, D.F. Hendry & N. Shephard, 'Computationally Intensive Econometrics Using a Distributed Matrix-Programming Language', Philosophical Trans. of the Royal Society A, vol. 360, pp. 1245-1266, 2002.

[13] J.L. Tripp, M.B. Gokhale & A. Hansson, 'A Case Study of Hardware/Software Partitioning of Traffic Simulation on the Cray XD1', IEEE Trans. on VLSI Systems, vol. 16, pp. 66-74, 2008.

[14] W.-c. Feng, 'Making a Case for Efficient Supercomputing', Queue, vol. 1, pp. 54-64, 2003.

[15] D. Chen, L. Wang, X. Wu, J. Chen, S.U. Khan, J. KoÇodziej, M. Tian, F. Huang & W. Liu, 'Hybrid Modelling and Simulation of Huge Crowd Over a Hierarchical Grid Architecture', Future Generation Computer Systems, 2012.

[16] J. Decraene, Y.Y. Cheng, M.Y. Hean Low, S. Zhou, W. Cai & C.S. Choo, 'Evolving Agent-Based Simulations in the Clouds', Int. Workshop on Advanced Computational Intelligence, pp. 244-249, 2010.

[17] H. Heng, L. Ruixuan, D. Xinhua, Z. Zhi & H. Hongmu, 'An Efficient and Secure Cloud-Based Distributed Simulation System', Applied Mathematics & Information Sciences, vol. 6, pp. 729-736, 2012.

[18] P. Wittek & X. Rubio-Campillo, 'Scalable Agent-Based Modelling with Cloud HPC Resources for Social Simulations', IEEE Int. Conf. on Cloud Computing Technology and Science (CloudCom), pp. 355-362, 2012.

[19] R.M. Fujimoto, 'Parallel Discrete Event Simulation', CACM, vol. 33, pp. 30-53, 1990.

[20] K. Venkatesh, T. Radhakrishnan & H.F. Li, 'Discrete Event Simulation in a Distributed System', IEEE COMPSAC, 1986.

[21] K.M.C.a.J. Misra, 'Distributed Simulation: A Case Study in Design and Verification of Distributed Programs', IEEE Trans. on Software Engineering, vol. SE-5, pp. 440-452, 1979.

[22] D.R. Jefferson, 'Virtual time', ACM Trans. on Program. Lang. Syst., vol. 7, pp. 404-425, 1985.

[23] O. Rihawi, Y. Secq & P. Mathieu, 'Synchronization Policies Impact in Distributed Agent-Based Simulation', AISC Distrib. Computing & Artificial Intelligence, vol. 217, pp. 19-26, 2013.