# JavaScript Distributed Agent Based Discrete Event Simulation

Daniel Zehe*, Heiko Aydt*, Michael Lees†and Alois Knoll‡
*TUM CREATE, 1 CREATE WAY, Singapore
Email: {Daniel.Zehe,Heiko.Aydt}@tum-create.edu.sg
†Nanyang Technological University, School of Computer Engineering, Singapore
Email: mhlees@ntu.edu.sg
‡Technische Universität München (TUM), Institute for Informatics, Robotics and Embedded Systems, Germany
Email: knoll@in.tum.de

***Abstract*—This paper shall give an insight into a novel idea of distributed agent based discrete event simulation. Instead of distributing work packages over different nodes of an high performance cluster, the simulation workload is distributed all over the Internet and the calculation is done in the JavaScript rendering engine on off-the-shelf computers, smart phones and other Internet connected devices capable of rendering websites and executing JavaScripts. An introduction into the general idea and the application possibilities for agent based simulation is given as well as a prototype application using this method is analytically and experimentally evaluated.**

## I. Introduction

The computing power for all kind of calculations increases steadily and a threshold is eventually reached. Right now the computing power on a single processor can not be increased anymore without intense heat dissipation. Therefore, manufacturers of processors tend towards multi- and many-core systems to keep up with the computational demands of computing tasks. This represents a form of distributed computation in a small scale of multiple cores within one processor die or multiple dies on one locally close motherboard. There are approaches towards even more locality distributed computing nodes like *X as a Service* (XaaS) [1]. Cloud computing initiative by several entities[1] have been really successful and are now accepted by industry and academia.

What would the computing power of the entire Internet and all its connected devices capable of rendering web pages be remains an open and non-trivial question. In order to assess the computation power, workstation or laptop computers, smartphones and tablets or other connected devices have to be accounted for. With roughly 2.41 billion Internet users around the world the global computing capacity which relies entirely on the capability of the web browsers rendering engine, can be estimated at around 723 PetaFLOPS. When assuming that each user has 300 MegaFLOPS of computing capability. Table I shows the Internet usage data [2] with the capability assumption made for the different continents[2].

This computing power is not fully used when browsing the web, because after the actual website is loaded and the graphics and texts are rendered the main workload for the processor is done. Depending on the type of website, there might be some JavaScript calculation or asynchronous data loading and sending using XMLHTTPRequests[3] (commonly know as AJAX). This leaves room for the calculation of other (possibly simulation) problems.

TABLE I. Internet statistic from June 2012 [2]

| continent | # of Internet user | population |
|---|---|---|
| Africa | 167,335,676 | 1,073,380,925 |
| Asia | 1,076,681,059 | 3,922,066,987 |
| Europe | 518,512,109 | 820,918,446 |
| Middle East | 90,000,455 | 223,608,203 |
| North America | 273,785,413 | 348,280,154 |
| Latin America | 254,915,74 | 593,688,638 |
| Oceania | 24,287,919 | 35,903,569 |
| **All** | **2,405,518,376** | **7,017,846,922** |

Simulation efforts in any field have one fundamental problem. When you increase either the number of simulation steps, hence making the simulation more granular by computing more logical time steps in a time stepped discrete event simulation, the computational power in terms of hardware resources necessary needs to be increased by the same amount. Also, with agent based simulation when increasing the number of agents per environment the scale of the simulation increases as well. This means not only the data for all agents increases but because the state of one agent depends on the state of other agents the number of calculation increases as well. A single off-the-shelf workstation computers can not support a feasibly timed simulation like that. High performance computing (HPC) is a valid option towards speeding up simulation.

A genuine problem all computational intensive tasks are facing is the performance for price ratio. The computing power of the Internet is virtually free and not yet regulated by any authority. Of course there are regulations and laws that prohibit harmful execution of code or stealing information of a remote computer but since most web-based code is executed in a limited functionality sandbox the amount of private or confidential information accessible without the users knowledge is limited, given that the security management of the execution engine is well developed but this shall not be the topic of this paper. Aside from those regulative and governmental restrictions, if it is possible get instructions and payload data on a computer on the Internet why not use this computational power for agent based simulation problems? Sending an agents' state together

[1] e.g. Amazon EC2 , Google AppEngine, Microsoft Azure
[2] different stages of technological development are not accounted for
[3] http://www.w3.org/TR/XMLHttpRequest/

with the necessary environment information to a computing entity (web site user), together with the instructions on what to do with the payload data will (eventually) return the right results for one time step of a simulation.

The first part of this paper contains background information on distributed discrete event simulation and a literature review of other peoples work necessary to understand the following techniques and assumptions made in the rest of this paper. Continuing with a general system design and a prototype implementation of a simple agent based simulation. Afterwards, an analytical and experimental analysis of the capabilities and boundaries of such system is made, before discussing challenges and concluding with a outlook towards the future of this technique.

## II. Related Work

### A. Distributed Agent Based Modeling

Agent-based modeling and simulation has grown in the last couple of years and matured into its own discipline for conducting research and some even say its a third way of reasoning besides inductive and deductive reasoning [3]. The opportunity of having complex systems described by the actions of autonomously acting objects called *agents* is exciting and a fair amount of research went into the different fields of application and the underlaying foundation [4]. From a modeling perspective an agent has to be (a) uniquely identifiable among other agents of the same kind that (b) (re)acts on its environment, (c) makes decision (only) on the perception of the environment accessible to it and is able to (d) change its rules and consequently its behavior on its own when the environmental conditions call for it [3]. All these functions have to be available for an agent in order to achieve its goals determined by the purpose of a given simulation. Agent based modeling and simulation is widely used in traffic simulations [5], behavioral sciences [6], biology [7] and research areas dealing with complex adaptive systems [8].

One of the earliest agent based models can be found by looking at *Reynolds* research on the distributed behavioral model of herds, flocks and schools from 1987 [9] explaining that three simple rules can be used to describe the flocking behavior of birds in a simulation.

Parallelization and subsequently distribution over a differently located network of computation nodes is a logical consequence for such autonomously acting agents. The state of agents and the environment is usually transfered using a message passing interface (MPI) when distributing within a locally closed environment. In order to parallelize the simulation workload efficiently a partitioning strategy has either be chosen statically based on locality principals or changed dynamically when the situation of the simulation calls for a more efficient execution mode of the workload.

Not only the partitioning of the workload poses a problem for large scale agent based models but also the distribution of the workload. There has to be a framework of describing the agents and the environment in addition to distribute the workload feasibly. Such system should be as generic as possible and provide the possibilities to be used in many fields of research and industry.

There are tools and frameworks in the academic roam such as the HLA_AGENT described by Lees et al. [10] or the *DGensim* distributed timeshared simulator for multi-agent systems proposed by Anderson [11]. The system by Anderson uses a centralized environment manager and remote (computing) nodes in order to distribute a certain number of agents to a physically different computation machine. This kind of simulation system usually uses the *High Level Architecture* (HLA) introduced by the department of defense of the USA [12] for transporting information between nodes. HLA can also be used to consolidate multiple agent-based simulations into one global simulation environment and have implicit distribution of workload as proposed by Scerri et al. [13]. Their approach also relies on a centralized environment service and remote working units.

In order to exchange information necessary for the correct execution of the simulation step different interest management schemes have been devised over the years [14]. The problem that the communication overhead to all computing nodes can be substantial has been addressed. Filtering techniques based on multicast subnet addressing or HLA filtering is effective but no additional communication while executing a simulation step would be even better but bares it own complications. But even only communicating at one point can be complicated due to synchronization methodology used and the management overhead of deadlock resolution or reissuing of falsely calculated simulation steps [15]. Each node has to assume that the environment information provided by the server is correct and sufficient to rightfully calculate the given time step. Methods like optimistic and conservative synchronization[16] can be used to widen the number of simulation steps that can be calculated before a resynchronization with the server is necessary. This also generates a management overhead at the server side which can lead to a price increase for operating such system.

### B. Distributed JavaScript Computing

Web services and web based technologies have very much grown over the last couple of years and the capability of a web browsers' rendering engines and JavaScript engines have evolved very fast. The performance measured on the *SunSpider JavaScript Benchmark*[4] increased across browsers by one order of magnitude [17] during the last decade. In addition to the bare performance increase, the technologies surrounding Internet computation and web design has changed as well. Beginning from HMTL 4 to HTML 5 many features were included and this makes all applications which are essentially websites look and behave almost the same on different operating system and browser platforms. These features like the *XMLHTTPRequest* used for *Asynchronous JavaScript and XML* (AJAX) requests and very recently the introduction of *WebWorkers*, a way to thread a JavaScript decoupled from the rendering of the actual web content, opens ways of using the capabilities of web browsers for more than just rendering of web pages.

The behavior of Internet users has also changed over the last two decades. Starting from the early nineties with

[4] http://www.webkit.org/perf/sunspider-0.9/sunspider-driver.html

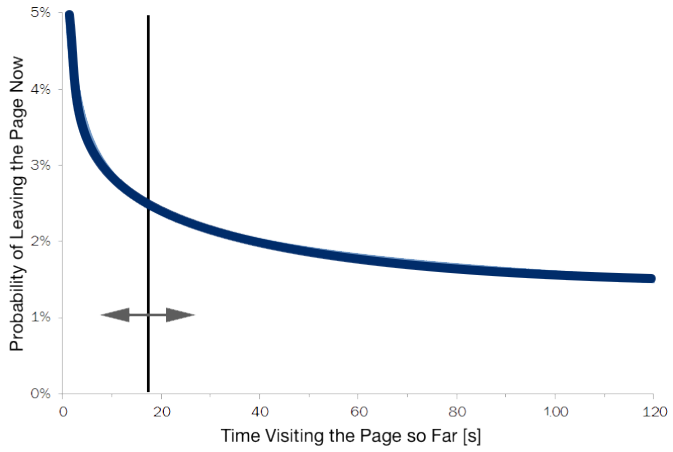


Fig. 1. Probability of leaving a website depending on the time already spent [1]

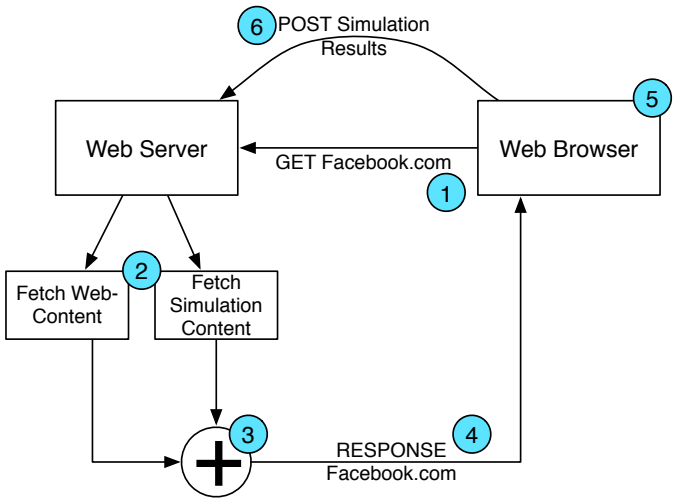


Fig. 2. The six stages of interaction and result calculation

dial up connections and per minute charging up to unlimited fiber-to-the-home connections available today the browsing behavior has changed dramatically. Where people tried to get information from websites as fast as possible just a couple of years back, the rich and engaging websites and social interaction possibilities encourages users to stay longer on a certain website. Lui et al. conducted a study how the content and the structure of a website encourage users to stay longer and how the possibility of leaving a website changes over time [1]. This information is useful in order to determine the possible target audience and website types for systems such as the one described in this paper.

The graph in Figure 1 shows the probability of users leaving a site depending on the time already spent. The black line which is just a rough estimate on where to engage with the described system. Left of the line the actual page content might not be fully loaded or the user already left and for longer calculation one must be sure that the user does not leave the website while the calculation is still in progress. Therefore, the simulation calculation should start after that imaginary line to the right.

Attempts to use the computing power of distributed JavaScript applications have been introduced before. There are systems testing and breaking MD5 salted cryptographic hashes[5] and also a MapReduce application called *MapRejuice*[6] but both systems are no longer developed or maintained.

## III. System Design

How should a simulation system using web pages as a the executing entity for a time stepped discrete event simulation look like will be introduced in this section. The technical parameters, ideas and implementation challenges that make such a system an efficient way to compute agents behavior and interactions with the environment will be discussed. Starting from a general overview of the system design and extending this with a prototype implementation of the Boids flocking behavior simulation discussed by *Reynolds* [9].

[5]http://www.andlabs.org/tools/ravan/ravan.html
[6]http://maprejuice.com

### A. General System Design

The general idea is to have a two part system composed of a normal web server and a number of computing nodes in form of web browsers. This server delivers web pages to Internet users accessing a specific domain but also holds the information and source code for a holistic or partial simulation problem.

Containing the environment information consisting of (a) the setting and size of the environment and (b)global parameters necessary to ensure correct calculations the server becomes not more than a global environment hub. The actual agents (agent state) and runtime information about the simulation problem are delivered in addition to the actual content of the web page requested by the user. This payload data is required by the clients to calculate the succeeding time step. The payload data consists of the agent state that is to be changed and its environment (other agents of the same or different kind or static obstacles).

The actual computation is then done in the JavaScript engine of the browser accessing the web page. This is a difference from an object oriented approach where the data and operations on the data are seen as one unit. With the introduction of new technologies related to the *HTML5* specification this can even be done in threads (WebWorkers) apart from the actual rendering of the front-end website [18]. The background calculation puts no distraction or latency on the user of a web page and runs almost transparent (unnoticed) of the user.

Figure 2 shows the general overview of the interaction between a web browser and the web server. The interaction can be broken into the following six steps.

1) Send a GET request towards a specific domain on the Internet (e.g. Facebook.com) either as initial request or a an interaction with the site. This will be subsequently called *Page Interaction* (PI).
2) The web server handles the request as a normal request but in addition to fetching the necessary HTML, CSS, and JavaScript files, it also fetches a JavaScript file with the simulation instructions as well

as the payload data.

3) Before returning the web page payload back to the users browser it is combined with the simulation instructions and payload.
4) The response is sent to the web browser
5) The web-browser renders the web page as usual and additionally calculates the simulation result in a separate (worker)thread.
6) After finishing the calculations, the result is sent back using POST messages to the server and incorporated into the global simulation environment.

After all agents necessary for one time step are processed and sent back to the server, the global environment is updated. This means all agents can be distributed to clients for an upcoming time step. Because web browsers of different clients can not communicate with each other as it is done in other distributed computing environments via MPI messages or HLA, the whole work package has to be self contained. Therefore, all necessary information has to be included in order to compute the next step correctly without having dependencies to other agents at the same time step. The difference between this approach of distributing the workload to clients (web pages) in comparison to a dedicated application approach is that no additional software needs to be installed and still providing the computational availability of an huge install base like BOINC or Folding@Home. The system behaves almost the same in respect to asynchronous execution of jobs after they have been dispatched by the server. No connections have to be held open and no HTTP-Sessions can expire, only a re-dispatch by the server can be initiated when there is no response after a certain interval.

There is also the inherent problem of the same origin policy enforced by many browsers. This hinders websites and also scripts to load and send data to any other location than on the same domain as the original website [19]. However, since advertisement networks have the same problem delivering their advertisements to the website this problem can be solved through configuring the web server correctly or using HTML components especially designed for such applications (e.g. <embed>).

### B. Boids Prototype Implementation

The Boids flocking simulation introduced by Reynolds is a simplistic behavioral animation with reasonable implementation challenges. It describes in a dynamic way how a number of birds behave when flying in open space. The agents in this simulation are called *Boids*. They only adhere to three basic rules and still behave dynamically and at random.

1) *Collision Avoidance* – avoidance of colliding with other flockmates or environment objects in the surrounding area
2) *Velocity Matching* – attempting to match the velocity of other flockmates
3) *Flock Centering* – attempting to stay close to all other flockmates

There are some additional rules like the tendency towards a specific point, the fleeing from a given pray or to achieve a greater objective, these can be implemented but the basic rules already cover the general behavior of birds forming a flock in open space. As each boids' next position only depends its own current position, its current velocity and the properties of its surrounding boids in a specified area there is ideally no communication necessary and in this system not even possible.

We made some assumptions about the general framework of the simulation and this is important for a good and practical implementation of the system. Each boid is surrounded by a limited area it has observatory knowledge about called perception range $s_p$. Based on this knowledge the decision for movement is made. Meaning, that only a portion of the whole environment has to be sent as payload to the client.

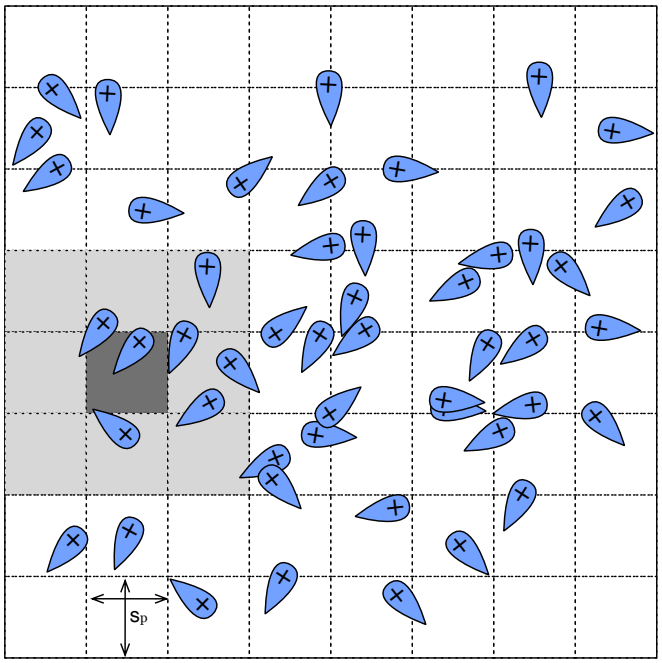


Fig. 3. Environment partitioned using perception range ($s_p$) information

Figure 3 illustrates how the whole environment is situated with arbitrarily distributed boids. Disregarding the dotted lines the partitioning pattern distributes the entire environment, for each agent that to a respective client for computation even though only a portion (in perception rage $s_p$) is used for the calculation of the succeeding position.

Regarding Figure 3 with the dotted line, a partitioning scheme based on the perception range $s_p$ of each boid is used. The size of one quadrant is exactly one perception distance in either dimension. In our implementation the perception is equal in all direction but other shapes or ratios are possible. Therefore, only the surrounding boids that could be reached have to be transfered over the network. The state of the whole environment depends only on the state of all boids. This way only the properties of the boids have to be transfered to a client.

In our implementation an extended markup language (XML) based exchange format for transferring the state consisting of location and velocity for two dimensions is used. The location does not consist of two parameter but rather of four. The first two are the $x$ and $y$ numbering of the quadrant

the boid is located in the global environment and the second one is the relative location within this quadrant. This allows for a easy transformation from a global location known only to the server to a local environment build by the executing client. Also, for the environment information for each quadrant the location relative to the center quadrant (dark grey in Figure 3) is transferred. This is necessary to reconstruct the local environment on the client side.

The actual calculations of all agents within one quadrant are done on one client. This reduces the number of environment transfers because for each boid within one quadrant to one transmission.

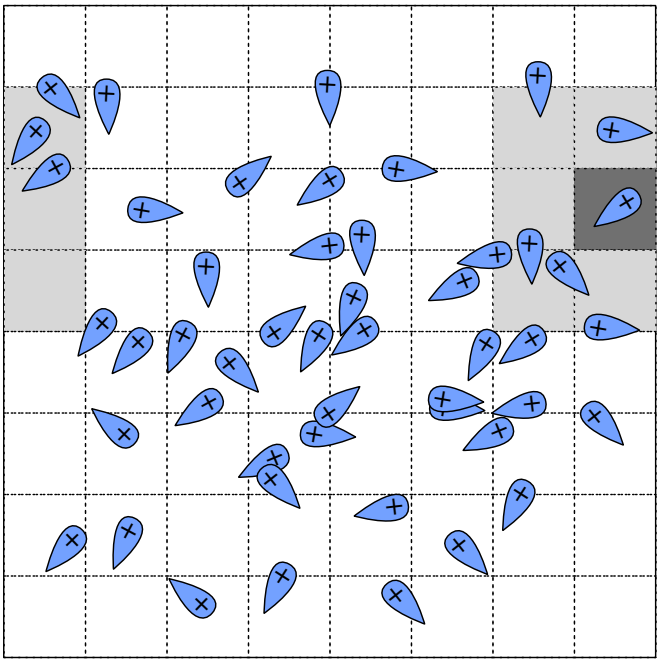

Fig. 4. Global environment layout with globe like mapping

On the HTTP environment server side the boids are held in a two dimensional data structure with a list holding all boids for each quadrant. The two dimensions are the $x$ and $y$ values for the quadrants. This data structure allows that on dispatching a new quadrant for execution not all boids have to be iterated through but rather just take all boids within one quadrant with a constant access times. Because the data structure is accessed through the quadrant information all returning boids from the clients can easily be assigned to their respective location.

The environment is also set up in a way that boids on the edges of the actual environment can also perceive other boids on the opposite edge. This way the environment seems endless like mapped on to a globe (see Figure 4). Because the server sends the environment information out with the relative location to the center quadrant the client does not know where on the global environment the payload boids are situated, just on a local environment. This makes it really easy to map the global environment on the server to the local environment at each client.

The synchronization on the environment server is done after every time step. Therefore, the server holds a shadow data structure of all finished quadrants and replaces the one used for dispatching after all quadrants are computed. This makes it also easy to re-dispatch a quadrant if there is no response within a certain interval after dispatch. Such a scenario can occur when a user closes the webpage the calculation is running on before the calculation is complete, the Internet connection is poor or the server is reachable for whatever arbitrary reason. The re-dispatch technique can also be used to verify results in case inconsistencies are recognized. Since we only implemented one layer of shadow data structure the consistency check has to be done before every synchronization.

## IV. Evaluation

The experimental evaluation of the Boid implementation will give some performance measures regarding latencies and networking overhead and will provide insights and possibilities for optimization.

### A. Analytical Evaluation

The analytical evaluation will focus on the obvious problems of critical path for the fastest time one simulation step can be executed. Other calculations such as partitioning optimizations and limitations towards the currently used methods are shown. The performance in respect to how long it takes to calculate one simulation step for the entire environment is strongly dependent on factors like the individual performance of each client, the network connection, the occupancy of the server, lastly all those go into the calculation of the critical path. Assuming the connected clients perform all with the same maximum computing power the actual execution time of the simulation step at 100% parallelization is fixed. A bottleneck for the critical path is created through the fact that not all clients have the same network conditions. The slowest network connection therefore determines the critical path. As it can be seen in Figure 5 the critical path is not necessarily be determined by the last client executing a simulation calculation but rather by the last result that returns back to the server ($t_{n-1}$). Because there can not be any calculation for the succeeding time step before all quadrants have been processed (without specific domain knowledge) the minimal execution time is then given by the maximal time ($t_{min} = max(t_i)$) of a round trip plus the actual execution time at the client $i$.

If we change the assumption even more and take the degree of parallelization and the different computing capabilities into account the critical path is even longer and more nondeterministic. The actual execution time $t_{all}$ of one time step is between the above given estimate and the worse possible solution $t_{max}$ where each client only requests a working package after the proceeding one is done or even worse when there is no package requested for some time $x$ at all.

$$t_{min} < t_{all} < t_{max} \tag{1}$$

$$t_{max} = \sum_{i=1}^{n} t_i + x \tag{2}$$

Heuristics and statistical evaluation of a real running system could serve as a benchmark for the occupancy and performance of such a distributed system.

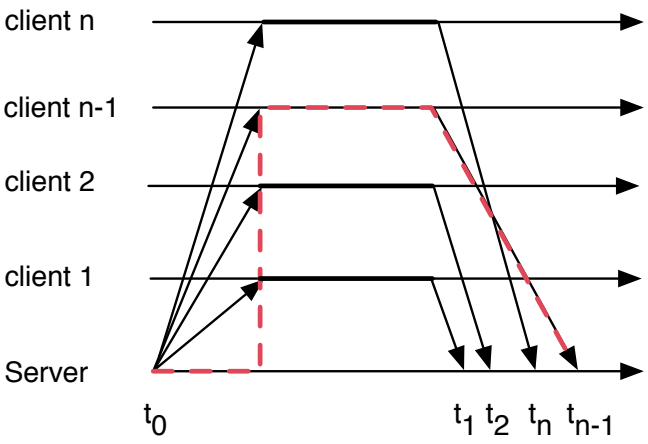


Fig. 5. Critical path for 100 per cent parallelized calculation

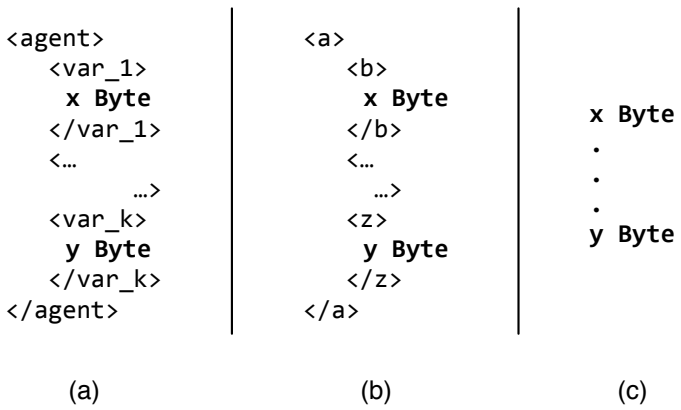


Fig. 6. Comparison of payload sizes for state transfer

Partitioning of the workload to a multitude of clients is a tough decision and static partitioning might not be the best option. In our example implementation we decided to partition the working on the quadrants as described above. While probably being the best choice when the environment is uniformly filled with agents, as soon as some quadrants gets congested and many agents occupy one partition the load is not distributed anymore. But for a agent based simulation, almost all agents have a physical dimension thus limiting the number of agents per partition to a maximum and at least some parallelization is possible. In the worst case all calculation are done on one client what has in best case the same performance than the serving machine itself.

The transfer of workload and environment information from the server to the client can be done in various ways. Since agents can be considered as objects in the programming sense, an object exchange format like XML or *JavaScript Object Notation* (JSON) can be used. The advantages such formats is they are easily human readable and supported by many platforms and operating systems. This also inherits a problem of transferring more data for each agent and its environment. For each variable the opening and closing tag has to be accounted for and in order to separate multiple agents, corresponding opening and closing tags have to be used as well. The additional meta-information might even be larger than the actual payload the XML-tag holds.

If the used data structure for an agent is known, a byte based data transfer might be the best option of reducing the overhead of structuring the transmitted data but reduces readability and flexibility. Figure 6 shows all three alternatives. The payload for the most human readable version (Figure 6(a) is the sum of the characters for the object name $N_o$ twice plus the characters for the variable name $N_v$ twice plus the brackets and slashes (5 for each $N$)and the number of actual variables $V$. This makes $N_{all}$ additional characters to the payload information.

$$N_{all} = 2N_o + 2N_v + 5(\#V + 1) \tag{3}$$

For the option shown in Figure 6(b) the assumption that there are no more variables than possible characters minus one in the text encoding used. For a UTF8 (1 Byte per character) encoding this would leave room for 254 different variables and one for the separation of the agents. This would result in $N_{all}$ characters in addition to the payload information.

$$N_{all} = 7(\#V + 1) \tag{4}$$

The third transport form shown in Figure 6c is not humanly readable at all. Nevertheless this transport form has an advantage that only in the payload information is transmitted. Changes in data type that result in more or less bytes to be interpreted on the receiver end would also result in a malfunction of the calculation.

The networking overhead regarding the amount of data that has to be transfered has one additional component that needs to be considered – the number of total communications between the server and the clients. This depends highly on the chosen partitioning strategy. If the partitioning strategy is spatial partitioning then the communication count is the total number of partitions ($n$) possible less the spaces not occupied ($k$) by at least one agent. Because the payload and instructions are transmitted when a website is requested no extra connections have to be made. Therefore, we call the number of outgoing connections $n - k$ and the number of incoming connections to the server $m$. The incoming connections can be modeled in the way that each agent in one partition returns their results separately or all are returned at one. Because the "return at once" strategy could result in additional overhead the first option is chosen and $m =$ total number of agents. This leaves us with the number of connections $c = (n - k) + m$ because for each time step all occupied partition have to me send out and all agents have to send their respective results back to the server.

### B. Experimental Evaluation

For the boids example implementation discussed in section III-B we designed an experiment measuring the execution time a client needs to calculate one simulation step for all boids within one quadrant. Therefore, the actual complexity can vary depending on the number of boids per quadrant.

The calculation consists of calculating the next position of a boid regarding the above (Section III-B) described rules. As the simulation evolves the boids move closer together and the formerly uniformly distributed set of boids flocks

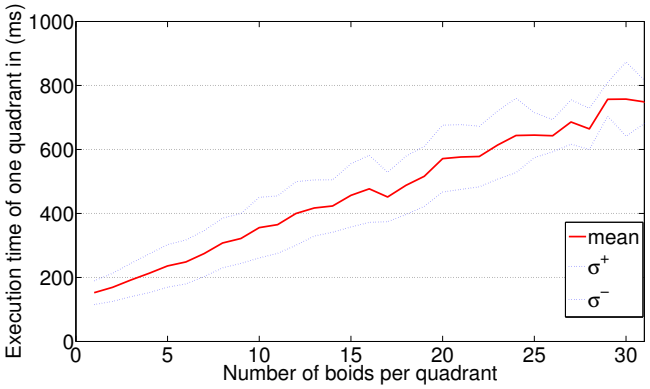


Fig. 7. Development of boids per quadrant the the respective execution time for the calculation of the quadrant

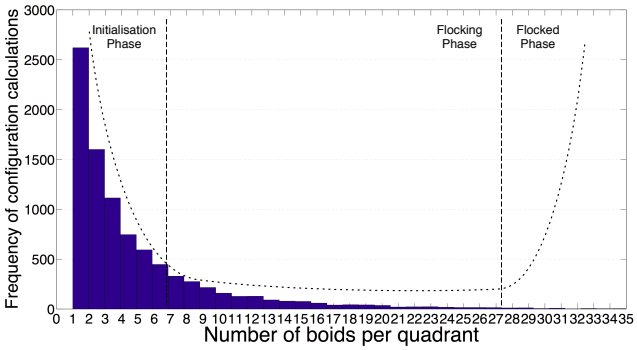


Fig. 8. Amount of times n boids had to be calculated by a total of 9000 calculations

together only covering a portion of quadrants. Figure 7 shows the development for an evolving simulation regarding the execution time in dependency to the number of boids per quadrant. It can be seen that there is an initial overhead for building the local environment (roughly 100ms) but after this is accounted for the increase in execution time by increasing the number of boids per quadrant is linear. The margin of error between the mean values displayed in the red line and its minimum and maximum values is bigger in the lower numbers because the amount of times this calculation occurred during all calculations is higher. Whereas the peak values are lower in the often calculated quadrant configurations the standard deviation is small.

This is shown in Figure 8. There, it can be seen that it is very common that only a few boids are within one quadrant. Because in the beginning of the simulation all boids are uniformly distributed over the given space each quadrant only contains small number of boids to process. When the simulation runs until a steady state has been reached and flocks have formed, there should also a sweet spot visible. A bathtub like curve with a high number of sparse quadrants (initialization phase) and a more or less constant number of transitional amounts (flocking phase) and a high amount of maximum possible boids per quadrant (flocked phase). This estimate is shown by the dotted curve in Figure 8 as well as the compartmentalization into the three described phases of flocking development.

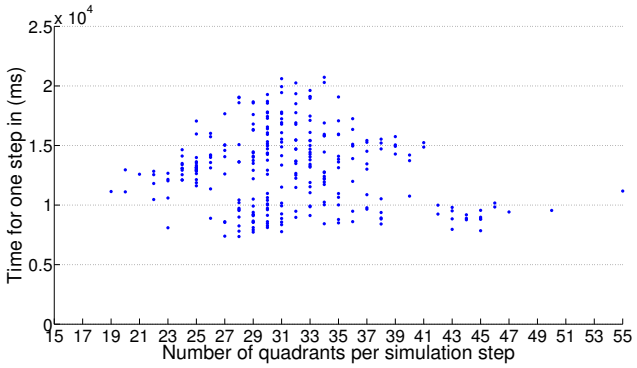


Fig. 9. Execution time for one simulation step depending on the total number of quadrants

In a second experiment we want to analyze the execution time for one simulation step and evaluate the influence of the increased networking and server side integration overhead. Figure 9 shows the total time $t_{all}$ necessary to execute one simulation step. The values are measured on the server and consist of the sequential execution time $t_{ex}$ for all boids (100) and the transfer time $t_t$ from and to the server and the integration time $t_{int}$ into the global environment.

$$t_{all} = t_{ex} + t_t + t_{int} \tag{5}$$

This illustrates that the overall execution time is very much dependent on the environment configuration of the time step. Because a lower number of quadrants would result in fewer actual transfers but the amount of data stays the same. There is a slight decrease in overall execution time when having more quadrants to distribute because the networking overhead is not as big in comparison to the payload data that is sent. Also the number of different variations possible when having 100 quadrants is smaller. This results also from the fact that on the flocked state of the simulation boids the computed quadrants are close together and sent more often as environment information of other quadrants than for calculation as when the quadrants are more sparse. The sometimes significant deviation from the mean values are due to the fact that the distribution within the given grid is not always the same therefore more or less boids belonging to the environment data had to be transmitted.
An example can be see in Figure 10 where the same amount (11) of quadrants are occupied by all boids (75). When considering the total number of boids that have to be transmitted as environment $E$ for on time step to all computing units as the sum of all environment boids for each quadrant $E_i$ then you end up needing less for a more sparse (unflocked) environment. In Figure 10(a) where the boids are more flocked the number of transmitted boids for this simulation step is 397 in addition to the 75 payload boids in the center quadrant. The number of environment boids in Figure 10(b) is only 165.

## V. Discussion

In this section we want to discuss the evaluation results and possible drawbacks, advantages or obstacle in deploying such system.

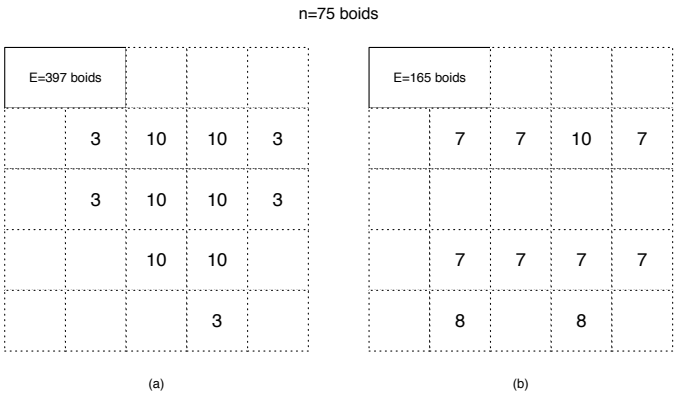


Fig. 10. Different sizes in environment data that has to be sent independent of the occupied quadrants

The results from the example implementation show that this system using the Boids example is easily scalable due to the linear execution time when more agents are present at one quadrant and have subsequently to be computed at one client. One advantage is that the barrier to entry to get a critical mass of computation nodes for this system is very low. The user does not need to know anything about the system. Big websites like facebook or google might already be doing calculations in the background without the consent of their users. Where other systems like BOINC [11] and Folding@Home [20] need user to install and maintain a piece of software on their machine this could run virtually on everything equipped with a modern web browser. From a workstation or Laptop computer to a smart phone or Internet equipped smart TVs.

Of course due to network latency and the scripting language JacaScript the performance is not comparable with real time HPC system but the pricing difference *Free vs. not-Free* might be an option. Also, if the distribution and partitioning is solved the number of possible clients and for that care computing nodes is greater than any super computer. The actual problem size sometimes does not require a for thousand of computing nodes and such system would offer a greater computing power then the problem can be broken down into or the distribution overhead would be to big.

An obstacle for such systems is to reach a critical mass. This can be achieved in at least by two ways. Involving the high throughput website administrators and their company to allow their users to be used as a computing node could be one solution. A additional solution would involve advertisement networks which are mostly stretched over a variety of websites and offer even a greater user base. This leads to the problem of compensating either the users or the website owners. There could be a model where the website owner gets paid for including the code in the website. But this is not the call to be made by the administrators because they are mostly using resources (Hardware and electrical power) of users. Therefore, the better system would be to pay the user who is willing to allow such computation to be done on their computer. This has the same problem as others like BOINC or Folding@Home have – commitment by the user. The option would be to still go to the website owners and have it as a second revenue model apart from advertisement. Offering user either to choose to contribute their computing power for a good simulation cause and having no advertisement or seeing advertisement.

The idea to have this kind of distributed simulation using JavaScript on client computers is an idea worth pursuing further because the agents are very easily assignable to clients and their state transferable to one client is quite easy if the information exchange between agents can be abstracted between steps. Besides the easiness of transferring agents from the centralized dispatching server to a web-page using serialization technologies like XML or JSON the amount of data as described in Section IV-A, the amount of data for large models with a lot of state information is significant. There should also be a model description language describing the models and sub-models to be present on each client and centralized dispatching server for the efficient updating the environment after each step. Also there are challenges and optimization possibilities related to geographical distribution and multilayer partitioning using cloud services as an intermediate layer between the actual simulation server and the end user clients. For load balancing or network traffic optimization this is inalienable. A cloud based multilayer approach with sub-environments might also be necessary for getting the load off the main environment server for scalability reasons but also introduces an additional layer of synchronization overhead.

## VI. CONCLUSION AND FUTURE WORK

Future research and engineering work topics should be concerned with extending the system design to be more simulation domain specific and allow a broader simulational functions. Also the monetization and acceptance study towards the willingness of internet users, who are the ones dedicating their computers for the cause on the one hand and on the other hand website providers with a wide enough reach to attract such users. The Usage of ever emerging web technologies such as WebSockets and also WebRTC for additional communication between the server and the client for additional work packages or the use of vector based processing architectures like WebCL[7] or to some extend WebGL[8] could also be helpful to speed up complex simulational calculations.

## ACKNOWLEDGMENTS

This work was financially supported by the Singapore National Research Foundation under its Campus for Research Excellence And Technological Enterprise (CREATE) program.

## REFERENCES

[1] C. Liu, R. W. White, and S. Dumais, "Understanding web browsing behaviors through weibull analysis of dwell time," in *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, Geneva, Switzerland, 2010, pp. 379–386.

[2] M. M. Group. (2013, jan) World internet users statistics usage and world population stats. [Online]. Available: http://http://www.internetworldstats.com

[3] C. M. Macal and M. J. North, "Tutorial on agent-based modeling and simulation," in *Proceedings of the 37th conference on Winter simulation*, Orlando, Florida, 2005, pp. 2–15.

[4] E. Bonabeau, "Agent-based modeling: Methods and techniques for simulating human systems," *Proceedings of the National Academy of Sciences*, 2002.

[7] http://www.khronos.org/webcl

[8] http://www.khronos.org/webgl

[5] V. Ljubovic, "Traffic simulation using agent-based models," in *Information, Communication and Automation Technologies, 2009. ICAT 2009. XXII International Symposium on*, 2009, pp. 1–6.

[6] D. Helbing, I. Farkas, and T. Vicsek, "Simulating dynamical features of escape panic," *Nature*, vol. 407, no. 6803, pp. 487–490, 2000.

[7] R. A. Hammond, "Peer reviewed: complex systems modeling for obesity research," *Preventing chronic disease*, vol. 6, no. 3, 2009.

[8] S. C. Bankes and S. C. Scientist, "Tools and techniques for developing policies for complex and uncertain systems," in *Proceedings of the National Academy of Sciences, Colloquium*, 2002, pp. 7263–7266.

[9] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, New York, NY, USA, 1987, pp. 25–34.

[10] M. Lees, B. Logan, and G. Theodoropoulos, "Distributed simulation of agent-based systems with hla," *ACM Trans. Model. Comput. Simul.*, vol. 17, no. 3, 2007.

[11] D. P. Anderson, "Boinc: A system for public-resource computing and storage," in *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, Washington, DC, USA, 2004, pp. 4–10.

[12] J. S. Dahmann, R. M. Fujimoto, and R. M. Weatherly, "The department of defense high level architecture," in *Proceedings of the 29th conference on Winter simulation*, Washington, DC, USA, 1997, pp. 142–149.

[13] D. Scerri, A. Drogoul, S. Hickmott, and L. Padgham, "An architecture for modular distributed simulation with agent-based models," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*.

[14] L. Wang, S. Turner, and F. Wang, "Interest management in agent-based distributed simulations," in *Distributed Simulation and Real-Time Applications, 2003. Proceedings. Seventh IEEE International Symposium on*, 2003, pp. 20–27.

[15] R. M. Fujimoto, *Parallel and Distribution Simulation Systems*, 1st ed. John Wiley & Sons, Inc., 1999.

[16] X. Wang, S. Turner, M. Low, and B. P. Gan, "Optimistic synchronization in hla based distributed simulation," in *Parallel and Distributed Simulation, 2004. PADS 2004. 18th Workshop on*, 2004, pp. 123–130.

[17] F. Smedberg, "Performance analysis of javascript," Master's thesis, Linkping University, Department of Computer and Information Science, 2010.

[18] I. Green, *Web Workers: Multithreaded Programs in JavaScript*. O'Reilly Media, 2012.

[19] M. Shema, *Hacking Web Apps: Detecting and Preventing Web Application Security Problems*. Elsevier Science, 2012.

[20] S. M. Larson, C. D. Snow, M. Shirts, V. S. P, and V. S. Pande, "Folding@home and genome@home: Using distributed computing to tackle previously intractable problems in computational biology," 2009.