Available at www.sciencedirect.com

INFORMATION PROCESSING IN AGRICULTURE 3 (2016) 235–243

journal homepage: www.elsevier.com/locate/inpa

Extending JASAG with data processing techniques for speeding up agricultural simulation applications: A case study with Simugan

Mathias Longo [a,b,1], Mauricio Arroqui [c,d,*], Juan Rodriguez [c,b], Claudio Machado [c], Cristian Mateos [a,b,1], Alejandro Zunino [a,1]

[a] ISISTAN – CONICET. Tandil (B7001BBO), Buenos Aires, Argentina
[b] Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Argentina
[c] Facultad de Ciencias Veterinarias – UNICEN. Tandil (B7001BBO), Buenos Aires, Argentina
[d] Agencia Nacional de Promoción Científica y Tecnológica (ANPCyT), Argentina

ARTICLE INFO

Article history:
Received 20 May 2016
Received in revised form
8 August 2016
Accepted 13 September 2016
Available online 19 September 2016

Keywords:
Agricultural simulation applications
Grid Computing
Gridification
JASAG
Data processing
Simugan

ABSTRACT

Resource-intensive agricultural simulation applications have increased the need for *gridification tools* –i.e., software to transform and scale up the applications using Grid infrastructures–. Previous research has proposed JASAG, a generic gridification tool for agricultural applications, through which the performance of a whole-farm simulation application called Simugan improved considerably. However, JASAG still lacks proper support for efficiently exploiting Grid storage resources, causing significant delays for assembling and summarizing the generated data. In this application note, two different data processing techniques in the context of JASAG are presented to tackle this problem. Simugan was again employed to validate the benefits of these techniques. Experiments using data processing techniques show that the execution time of Simugan was accelerated by a factor of up to 34.34.

1. Introduction

Agricultural simulation applications (ASA) are tools to simulate diverse farming factors such as crop and livestock yields, soil organic carbon content, greenhouse gas emissions [11] and energy balance, among others. As pointed out in [6], to date, several agricultural simulation applications –e.g., APSIM, CropSys, DSSAT, SUCROS– have been developed.

Agricultural simulations are inherently climate-driven and subject to market uncertainties [19]. For example, the climate might affect pasture growth rate, while certain market conditions might lead to different economic outcomes. Taking into account these potential variabilities, experimentation in this context requires performing many simulation runs of the models being tested so that confident results are obtained [15]. In addition, the individual execution of such models via ASAs is a big CPU time consumer [2], particularly in presence of complex models. For these reasons, dealing with

* Corresponding author at: Facultad de Ciencias Veterinarias – UNICEN. Tandil (B7001BBO), Buenos Aires, Argentina.
E-mail address: marroqui@exa.unicen.edu.ar (M. Arroqui).
[1] Fax: +54 (249) 4385681.

Peer review under responsibility of China Agricultural University.
http://dx.doi.org/10.1016/j.inpa.2016.09.001

ASAs in practice requires lots of computational resources, and thus parallel/distributed computing techniques must be applied [26].

Grid Computing is a computing paradigm that arranges multiple heterogeneous computational resources for the purpose of solving resource-intensive problems [9]. Within a Grid [8], resources such as processing power, storage and network bandwidth are combined into a "virtual" and powerful supercomputer. Effectively exploiting the benefits of a Grid comes however at the expense of programming applications so that Grid resources are used. Thus, researchers have investigated *gridification tools* to make Grid-enabling applications easier, which automatically or semi-automatically transform single-computer application source codes to Grid-aware codes. Then, current ASAs are an example of applications that might benefit from gridification. However, there is no one-fits-all gridification tool capable of gridifying any kind of software [16], and thus the need of providing specialized gridification tools arises.

Motivated by this need, a gridification tool called Java Agricultural Simulation Applications Gridifier (JASAG) was built [2]. From a software design perspective, this tool produces Grid-aware application source codes by exploiting the parallelism implicitly present in the common application design/structure ASAs have. This parallelism-friendly structure manifests in the way the modeled entities (e.g., animal, soil, pasture, and cash crop) and the tasks/computations processing them via custom mathematical models (such as feeding animals, growing animals, moving animals, growing pasture, growing crop) interact during a simulation [2]. Therefore, JASAG focus on executing these tasks/computations in parallel across the computers of a Grid.

The practical utility of JASAG was illustrated by Grid-enabling the Simugan [14] agricultural simulation application. Simugan enables users to model and exercise whole-farm simulation scenarios. After gridification, experiments showed that the performance of Simugan increased dramatically, with speedup factors of up to 25 [2]. Therefore, with the Grid-enabled Simugan it is possible, for example, to build larger simulation scenarios in terms of entities and years of simulation, or increase the number of concurrent simulations running in the Grid from different users, thus accelerating experimentation in the area.

On the downside, the current version of JASAG focus on efficiently using available processing power to execute the tasks in a simulation, but does not exploit techniques to properly manage the data produced through a simulation run. As tasks can be processed by any computer of the Grid, partial simulation results end up scattered in every computer wherein simulation tasks are executed. For example, in an experimental scenario with Simugan consisting of 200 concurrent small simulations, 14 GB of biophysical and farm management results were generated, so the data collection to build the final simulation output is very time-consuming. The current implementation of JASAG queries all Grid computers to request the associated partial data from a single computer. As a result, when running the gridified Simugan it took more time building output results than actually executing simulation tasks.

Since JASAG aims at scaling up agricultural applications exhibiting a particular (but very common) software design, this drawback affects, in many agricultural experimental situations, the efficiency with which the results of a simulation step are stored and collected from Grid computers to be presented to the final user [2]. For this reason, the goal of this application note is to improve the data management module of JASAG in order to further improve the overall performance of targeted ASAs. For that purpose, we will take as a base well-known data management approaches from the operating systems area such as caching or data grouping techniques.

2. Materials and methods

2.1. *Simugan and JASAG*

Simugan is a whole-farm simulator oriented to assist the research, teaching and technology transfer of alternative beef cattle production systems [14,1]. This simulator bases all its simulation in *scenarios*, each containing initial values and conditional rules to manage a farm. Users might create, modify, save, retrieve or delete their own scenarios. As output, the simulator generates a spreadsheet containing all the information for further analysis. Simugan is accessible from regular Web browsers (http://simugan.vet.unicen.edu.ar/simfarm/) and Android phones [1]. To date, Simugan has been used both for educational (e.g., teaching courses) and research purposes (e.g., [4]). Another notable Simugan user is the National Institute of Agricultural Technology (INTA) of Argentina, a governmental agency in charge of promoting innovation and research in agriculture.

In Simugan, a simulation comprises two main elements: entities and tasks. Entities –such as an animal or pasture– represent the biophysical system. Entities have particular properties. For example, an animal might have the properties *pasture intake*, *live weight*, *live weight gain*, among others. In addition, some entities might be in container entities, such as herds, which basically groups other entities and also have their own properties. On the other hand, tasks describe the operations (computations) that can be done on the entities during a simulation. There are two types of tasks: property update tasks, which just update certain properties of an entity, and rule-driven tasks, which are able to move, delete or add entities from/to container entities. This two-element structure has been employed to guide the design and development of many ASAs. Examples are the models proposed in [14,20,10,13] as well as the frameworks in [21,12].

In Simugan, tasks are executed within a *simulation step*, or a fixed unit of time (minute, day, week or month). The default simulation step unit is a day. In each simulation step, entities might flow from one task to another until all modeled tasks are executed. For example, an animal entity is first updated by the *Intake* task and then it is updated by the *Growth* task, because an animal can grow only after it was fed. This also shows that there are *dependencies* between tasks, meaning that certain tasks can be executed provided other tasks have finished. In the previous example, the *Growth* task depends on the *Intake* task, and as such they are called property update dependent tasks.

Moreover, taking into account this two-element structure, JASAG considers all the dependencies between the tasks of an ASA, and tries to execute as many independent tasks as possible in parallel in a Grid. Following the previous example, to exploit parallelization in that case, a producer–consumer scheme is adopted, where the dependent task (*Growth* in the example) waits to *consume* the entity that is processing the independent task (*Intake* in the example). This is done for every independent task-dependent task relationship in the system. Furthermore, JASAG comes with three different parallel execution strategies [2] depending on the grade at which parallelism is exploited: (a) simulation-level parallelization, where entire simulations run concurrently as black boxes, (b) task flow parallelization, where simulations and independent tasks inside simulations run concurrently, (c) and data flow parallelization, where simulations, independent tasks inside simulations, and property update dependent tasks run concurrently. This is, the data flow strategy extends the task flow strategy with the possibility of running property update dependent tasks using the parallel strategy under the producer–consumer entity scheme.

When a simulation finishes, the generation of the output variable values begins (e.g., the predicted weight of animals), and for that purpose JASAG has a separate module called *Output Generator*. This module looks for the simulation data dispersed in the different Grid computers, and then summarizes all the data into a file. Particularly, Simugan formats the summarized data as a spreadsheet that the user can download. The spreadsheet has statistics not only about the different entities, but also about economic, productivity and environmental results. For example, Table 1 depicts an extract of the output generated by Simugan for a simulation. The table shows four days of the simulation, and four statistics about one of the paddocks belonging to the farm. The first column shows the simulation step. The second column describes the pasture mass per hectare. The third column shows the number of animals eating in the paddock that day. In the fourth column the sum of all the animal's weights is shown. The fifth column shows the rate at which pasture grows.

To store simulation data, JASAG uses a key-value NoSQL database [22], where the key is a unique identifier assigned to each entity in the simulation and for each simulation step. The value is the entity itself with all its attribute values. Thus, each entity is "replicated" in each simulation step, storing all the historic attribute values of that entity across the simulation. For example, attribute values of a *Cow* entity instance are first stored in the simulation step 1, then they are stored again in a different entry for step 2, and so on. Fig. 1 depicts an overview of how simulations are handled in JASAG, where a Grid infrastructure with a database distributed among two computers can be seen. Upon a new simulation submitted by a user arrives, the associated entities and tasks are processed and executed in the Grid computers. After that, the Output Generator collects the information of a specific entity by searching the distributed storage for the values of that entity across all simulation steps, which implies network communication.

For each of the columns in Table 1, the Output Generator has to query the Grid computers to collect the data. For example, to calculate the total number of animals grazing in a given paddock, the Output Generator looks for entities whose type is "animal" and belongs to the respective paddock and simulation in the whole database. The same process is done for every paddock and every simulation. In order to do that, the Output Generator makes a query with the required information (in this case, the paddock and simulation identifiers) to retrieve the respective entities. Then, this query is broadcasted to all the computers in the Grid. Finally, each computer returns a list with all the matching entities and the returned lists are merged. As a result, a list with all the entities, from all the simulation steps, belonging to the respective paddock and simulation is obtained. This process takes a long time to run not only because of network latencies, but also because many entities are queried more than once since they are needed for producing more than one output value.

In the single-computer version of Simugan [14] (the one not gridified), this process was rather quick as the database and the data summarization process run in the same computer. After gridification using the current version of JASAG, this turned out inefficient since queries are broadcasted to every computer in the Grid several times due to the lack of support for data caching. The following subsections detail the data techniques considered in this paper to improve the mechanics of the Output Generator module in JASAG.

2.1.1. *Last recently used (LRU) cache*

The temporal locality principle [5] is present in most operating systems (OS) and helps to deal with situations where it is needed to access the same piece of information several times within a given time period. In OSs, this principle is exploited to build caches [23], i.e., data structures for faster access to memory data in RAM or disk which might be needed shortly after. Based on this, we implemented an LRU (Least Recently Used) cache or in other words a data structure that temporarily stores some entities, and each new entity to be

Table 1 – Simugan: sample output.

Day	Pasture mass [kg MS/ha]	Total number of animals	Total animal weight [kg]	Pasture growing rate [kg MS/ha]
7/1/2015	4,275.792	1030	458,786.086	7.792
7/2/2015	4,242.972	1030	459,503.568	7.534
7/3/2015	4,210.161	1030	460,224.489	7.276
7/4/2015	4,177.649	1030	460,948.456	7.019
...	...	...	...	...

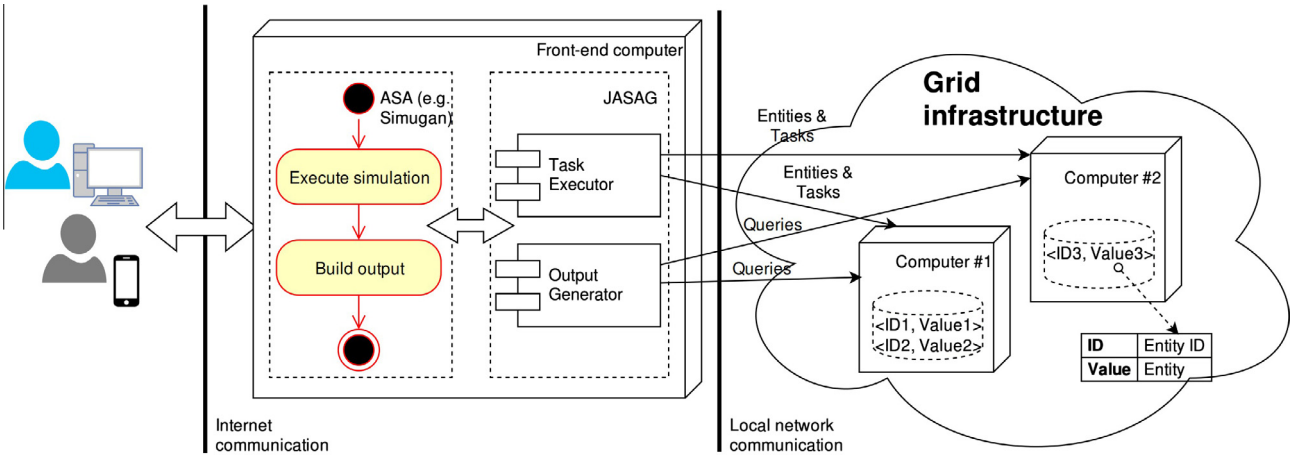


Fig. 1 – Running Simugan simulations via JASAG: overview.

stored replaces the least recently used in the cache. The LRU cache is implemented as an in-memory associative data structure located in the front-end computer.

Under this scheme, the Output Generator is again connected to the distributed database to which queries are made. But, upon making a query, the Output Generator first looks for the information in the cache, and if the information is not there, then it queries the database as it originally did. The results of the query are not only used in the calculations, but also temporarily stored in the LRU cache. As mentioned before, many entities are queried more than once, so when using the LRU cache those entities are temporarily cached and many queries through the network are avoided. Therefore, this approach should reduce the amount of queries made to the remote computers. As a result, the overall simulation time should be lowered.

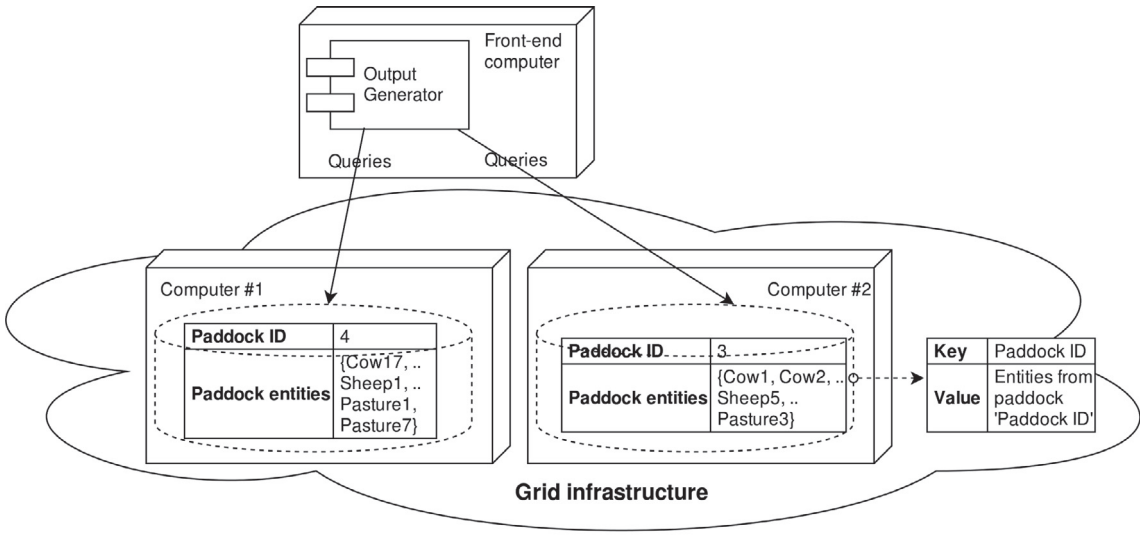
2.1.2. *LRU cache + Grouping*

As pointed out, the NoSQL database used by JASAG is a key-value database, where the key is unique for each entity created for each simulation. However, when the Output Generator analyses a group of entities, it has to retrieve each entity integrating that group separately. That involves a considerable number of remote queries, which is time consuming. Furthermore, the most used statistics in the spreadsheet are those involving group analyses (such as the total animal weight where all animal entities should be retrieved). In addition, this also leads to duplicated queries since an entity might be queried for performing different group analyses. In the previous example, all the statistics showed information about the group of animals or the pasture mass belonging to a paddock, in a particular simulation. Therefore, a possible improvement would be to group entities in the database by a certain pattern (container entity), and hence to make queries to retrieve groups of entities instead of separate, more granular entities. With this new improvement, less distributed queries might be needed.

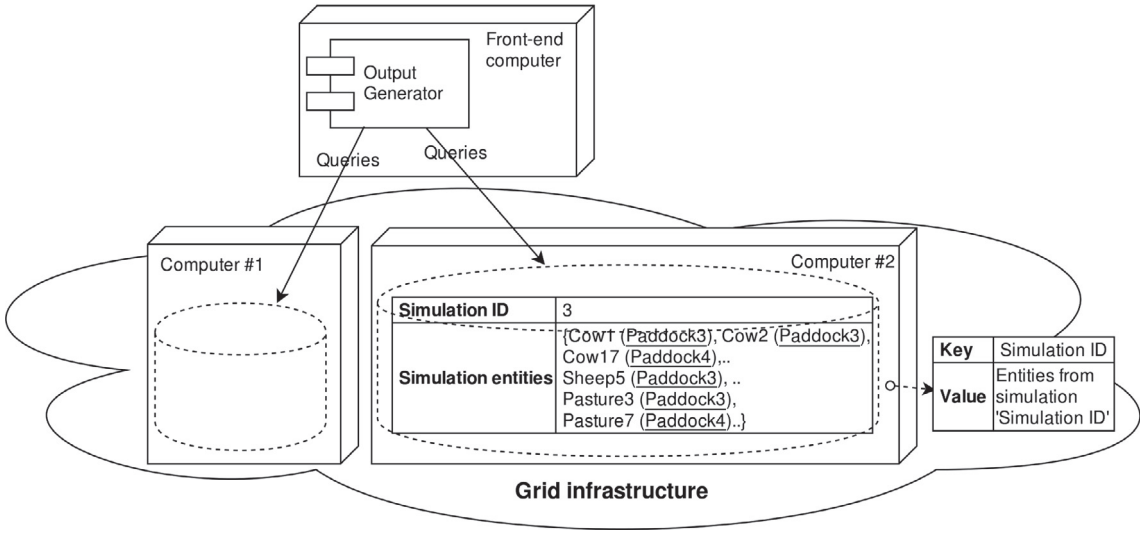
Several criteria could be considered for grouping entities depending on the root selected for the "entity tree" to return, such as group by simulation or group by a certain container entity, such as a *Paddock*. Based on Table 1, grouping by simulation means that the entire entity tree associated to a simulation is an entry in the distributed database mentioned in Section 2.1. If entities were instead grouped based on a lower-level container entity, all the entities corresponding to the paddock would be placed in an entry of the database, while the other paddocks would be in a separate group. Fig. 2 graphically shows the database using the mentioned grouping strategies. Fig. 2a shows entries grouped by paddocks whereas Fig. 2b shows entries grouped by simulation. Using either grouping alternatives, in the example, just one query would be needed to retrieve the entities associated to a specific paddock. In our implementation, the entities were grouped by simulation and as a consequence the key is the simulation identifier, because most of the analyses were made considering all the entities of a single simulation. This means that all the entities corresponding to a specific simulation were stored together in one entry of the distributed database. In JASAG, the grouping strategy can be nevertheless configured by the user.

So far, the described approach might however result in a trade off since there are cases in which it is not necessary to retrieve a whole group for a query. For example, if an economics major wants to analyze the results of a simulation, he/she only needs economic information, i.e., profits and expenses. Thus, grouping the entities requires retrieving information about animals as well as information about other paddocks that are not needed. In such cases, applying the grouping approach can make the output generation process slower than necessary.

Because of this trade off, this approach should be applied once the LRU strategy is implemented. Specifically, if the LRU strategy were not implemented, this approach would need to retrieve a whole group of entities each time a calculation is carried out. Considering Table 1, all the entities of the paddock would be queried for each column. Therefore, since the information is grouped in the database, each entity group that is queried is then temporarily stored in the LRU cache, thus there is no need to make queries to the database while the information is in the cache.



(a) Key-value database when grouping by paddock



(b) Key-value database when by simulation

Fig. 2 – Key-value database using grouping strategies.

2.2. *Experiments*

We used Simugan to evaluate the introduced data processing approaches into JASAG. Three versions of Simugan were considered: the gridified one as is (called Original), the LRU version, and the LRU cache + Grouping version. The simulation scenario was the same for all versions, and it consisted in a two-year simulation with more than 20 entities and 30 tasks. For comparison purposes, although most of the settings are the same used in [2], the time of simulation in this case is two years instead of 6 months, to produce more representative CPU usage and generated output data. Regarding the simulation load, i.e., the amount of simulations processed concurrently, initially we considered a 50-simulation load for the two implemented versions, while a 10-simulation load was considered for the Original version. This was done since the execution of the Original version took too much time to finish and the gains were enough for the analysis. Lastly, for this evaluation we used the data flow parallelization strategy (see Section 2.1), since it is the one which achieves the best speedups according to [2] for most scenarios.

After this, the application versions were run in an infrastructure with 48 cores distributed in eight computers, and a total RAM memory of 88 GB. Table 2 depicts the technical details of each computer. All computers run GridGain 6.5.6 (http://atlassian.gridgain.com/wiki/display/GG656/Home), a mature Java middleware that is currently used in conjunction with JASAG to provide connectivity between computers and data storage/access within the Grid. Additionally, the

Table 2 – Technical details of the execution environment.

Computer identifiers	CPU	Network controller	RAM	# of cores
1...5	AMD Phenom(tm) II X6 1055T (2.8 GHz)	Realtek Semiconductor Co., Ltd. RTL8111/8168B PCI Express Gigabit	8 GB	6
6...8	AMD FX(tm)-6100, Six-Core (3.6 GHz)	Qualcomm Atheros AR2417 Wireless Network Adapter	16 GB	6

Grid is configured so that the simulation data is uniformly distributed between all the computers during executions.

The metrics used for assessing Simugan performance included output generation time, overall execution time, speedup, memory usage, CPU usage, and the number of network packets transferred between computers. The overall execution time metric takes into account both the simulation time and the output generation time. The speedup metric measures how faster the version applying either of the presented data approaches (T_a) is versus the Original version (T_o): $S = T_o/T_a$. The CPU average usage metric was calculated from each computer core usage within 1-min periods, and then, these values were averaged. In the same way, the average memory load was calculated from each computer within 1-min period. Finally, the number of network packets is all the packets sent between computers exclusively during the output generation process.

3. Results

Table 3 shows all the metrics previously described. The first column contains the different versions of Simugan, i.e., Original, the one which uses the LRU cache, and the one using both the LRU cache and the entity grouping approach. Fig. 3 shows the time-related charts. Fig. 3a shows the execution time and the relation between the simulation time and the output generation time: the Original version took more than 200 min, whereas the *LRU* and *LRU + Grouping* versions took less than 30 min. Additionally, it can be seen that the relation between the simulation time and the output generation time decreases for the *LRU* and *LRU + Grouping* versions. Fig. 3b shows a speedup factor of 8 and 34 for the *LRU* and *LRU + Grouping* versions, respectively.

Fig. 4 shows the performance-related metrics. Fig. 4a shows the CPU usage of each version, where the Original version used nearly a 50% of the CPU, the *LRU* version used nearly a 40% of the CPU and the *LRU + Grouping* version used less than 25%. Fig. 4b shows the memory load, in which the Original and *LRU* versions had almost a 30% memory load, while the *LRU + Grouping* version had less than a 15% of memory load. Finally, Fig. 4c shows the amount of network packets transferred in the output generation process. The Original version sent more than 7500 packets during the simulation, while both the *LRU* and *LRU + Grouping* versions sent less than 2000 packets.

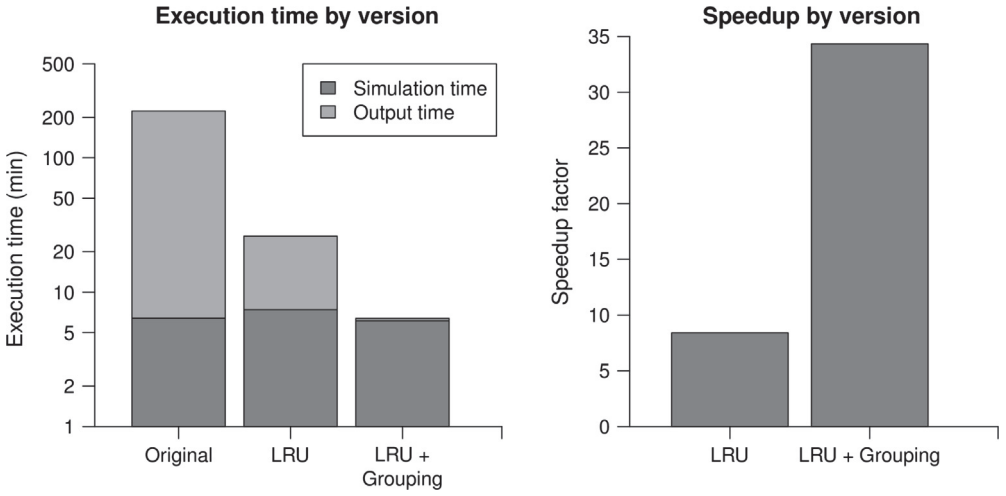
4. Discussion

The results obtained using the proposed data processing approaches show that the amount of packets sent through the network during the execution is considerably lower: the *Original* version sent nearly four times the amount of packets sent by the *LRU* version, and eleven times the amount sent by the *LRU + Grouping* version. This means that the initial hypothesis, by which few queries would be made by using either of the presented approaches, holds because there are less packets transferred. Therefore, the output generation is faster in both versions mainly due to this improvement, achieving very significant speedups: 8.41 and 34.34 respectively. This means that the *Original* version takes 8.41 times and 34.34 times the time needed to execute the *LRU* and *LRU + Grouping* versions, respectively. Additionally, such improvements did not result in more intensive use of CPU or memory. Furthermore, both metrics have similar and, in fact, lower values compared to the *Original* version. Since there are fewer queries due to the use of caching, there are fewer searches for entities into a large group of data in each computer. As a consequence, the CPU is used less than it originally was, and the memory is not as much loaded either. For the *LRU + Grouping* version, the entities are grouped, so it is no longer necessary to look for a specific entity as the search is for just one group.

Despite the fact that an execution with a 50-simulation load and an execution with 10-simulation load are not comparable, the goal of this work is not to strictly compare the execution times but to show that applying distributed data processing approaches the overall execution time would be improved. In that sense, a 10-simulation load was enough to prove it, and it is worth noting that if the *Original* version were executed with a 50-simulation load, the speedup would be substantially higher. Also, applying the presented strategies allows Simugan to run several simulations in the same time

Table 3 – Metrics summary.

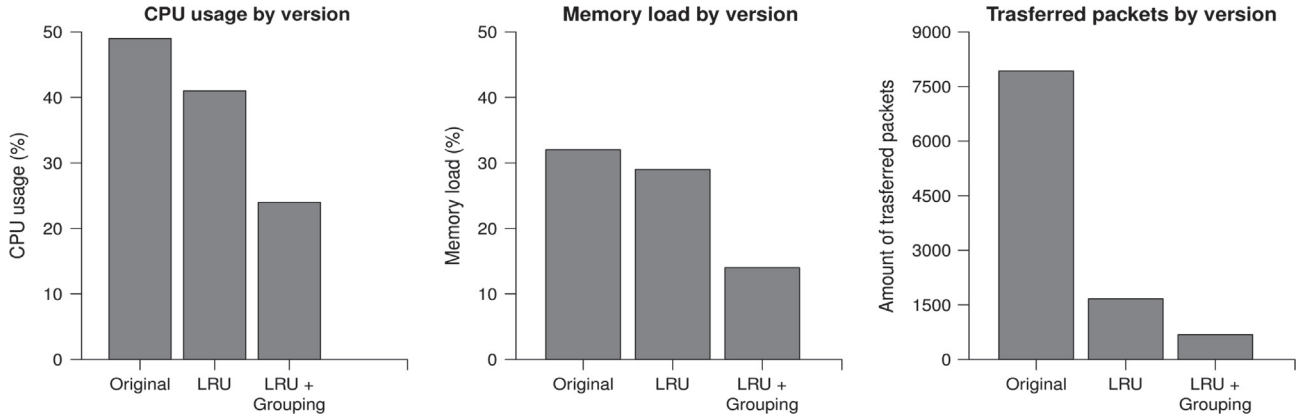
Simugan version	Output time (min)	Execution time (min)	Speedup factor	CPU usage (%)	Memory load (%)	Packets (qty)
Original	216.71	223.13	1.00	49	32	7925
LRU	18.70	26.10	8.41	41	29	1668
LRU + Grouping	0.26	6.39	34.34	24	14	680



(a) Logarithmic execution time chart for each version

(b) Speedup chart for the two enhanced Simugan versions

Fig. 3 – Execution time related charts.



(a) CPU usage

(b) Memory load

(c) Transferred packets

Fig. 4 – Performance related charts.

it took to run just 10 in the *Original* version, enabling the user to do a better domain analysis in less time. For example, if the user wants to find a scenario for a specific situation in the future, these improvements allow that user to try several scenarios with different configurations in less time it took before to test only one scenario.

To the best of our knowledge, there are no previous works applying data processing approaches to lower the overall execution time of agricultural simulation applications. However, there are works regarding gridification tools where similar approaches are used to lower the execution time of general-purpose distributed applications. In particular, these studies focus on replicating the data stored in distributed environments, so that the different computers do not have to query several times for the data. For instance, in [24] an LFU (Least Frequently Used) data cache was used for this purpose, decreasing execution times up to a 30%. A shared characteristic of these approaches is that they focus on file-system level data caching and replication, whereas we focus on *in-memory* data processing techniques. Both views, nevertheless, are complementary.

Moreover, in a previous work [26] another related whole-farm simulation tool –APSIM [13]– was gridified. APSIM is a well-known modular farm modeling framework, developed to simulate biophysical process in farming environments. The gridification approach in [26] is based on distributing tasks across a Grid as JASAG does. The gridification process was achieved by using the HTCondor [25] workload management platform. However, APSIM does not prescribe data processing approaches explicitly and, besides, HTCondor also focuses on managing data at the file-system level. That means that HTCondor uses the hard-disk drives in computers

to store simulation data, while in our approach the data is cached in main memory whenever possible, which speedups data access.

5. Conclusions

By borrowing and applying two memory administration techniques from the OS area to the problem at hand, we improved JASAG with data processing techniques, and the overall execution time of Simugan was lowered. The first technique is the LRU caching approach, where an auxiliary in-memory data structure was used to store the last recently used data entry (i.e., entities). With this structure, many network queries were avoided in the Output Generator module of JASAG and, as a result, the amount of time needed for the output generation phase when running simulations is lower. The second approach was the LRU + Grouping approach, where in addition to the LRU cache, the entities are grouped by certain domain-dependent composition patterns. In addition, as many of the statistics carried out during the output generation are done in groups, the amount of queries made to the Grid computers decreased and so did the output generation time.

Combining those approaches with ASAs led to not only a lower execution time, but also a lower CPU, memory and network usage. By applying the LRU cache, and thus exploiting the temporal locality principle, a speedup greater than 8 was achieved, and with both the LRU approach and the Grouping approach, we achieved speedup greater than 30. It is worth noting that these overall run time improvements offer users a better testbed to study a new spectrum of resource-intensive whole-farm scenarios, particularly those considering sustainability and climate change contexts. In these scenarios, the user of the agricultural simulation application should run at least 25 years of simulation and many experimental repetitions in order to obtain confident results [17,7]. Thus, the significant improvements achieved in this work for enabling this kind of experimentation in Simugan could open the door to other ASAs gridified with JASAG to do so.

As a corollary, when an agricultural simulation application is gridified, not only the execution time inherent to executing the model should be taken into account, but also the way the produced information is stored/retrieved to/from the infrastructure. Therefore, it is advisable to consider appropriate data processing techniques to improve the overall execution performance, and hence in this application note we have shown the benefits of in-memory data techniques in the context of JASAG in general and Simugan in particular. In this line, the performance and application-independence of JASAG makes it feasible to cast JASAG/Simugan as a generic fast whole-farm simulation "service" to be offered to external agricultural systems apart from users. In fact, there is currently a need to facilitate and automate agricultural systems integration [1,3], for which newer ICT technologies can be used. The Cloud Computing paradigm [18], which can be viewed as an evolution of the Grid Computing paradigm and comes with several service provisioning models (e.g., Software as a Service – SaaS), seems to be the right path to drive this research [3].

REFERENCES

[1] Arroqui M, Mateos C, Machado C, Zunino A. Restful web services improve the efficiency of data transfer of a whole-farm simulator accessed by android smartphones. Comput Electron Agric 2012;87:14–8.
[2] Arroqui M, Rodriguez Alvarez J, Vazquez H, Machado C, Mateos C, Zunino A. Jasag: a gridification tool for agricultural simulation applications. Concurrency Comput: Pract Exp 2015;27(17):4716–40.
[3] Barmpounakis S, Kaloxylos A, Groumas A, Katsikas L, Sarris V, Dimtsa K, Fournier F, Antoniou E, Alonistioti N, Wolfert S. Management and control applications in agriculture domain via a future internet business-to-business platform. Inf Process Agric 2015;2(1):51–63.
[4] Berger H. Modelling the effect of maize silage and oat winter forage crop on cow-calf systems in Argentina. In: International grassland conference; 2013. p. 15–19.
[5] Denning P. The locality principle. Commun ACM 2005;48 (7):19–24.
[6] Emmi L, Paredes-Madrid L, Ribeiro A, Pajares G, Gonzalez-de Santos P. Fleets of robots for precision agriculture: a simulation environment. Ind Robot: Int J 2013;40(1):41–58.
[7] Finger R, Lazzarotto P, Calanca P. Bio-economic assessment of climate change impacts on managed grassland production. Agric Syst 2010;103(9):666–74.
[8] Foster I, Kesselman C. The Grid 2: blueprint for a new computing infrastructure. In: The Elsevier series in grid computing; 2003.
[9] Foster I, Kesselman C, Tuecke S. The anatomy of the grid: enabling scalable virtual organizations. Int J High Performance Comput Appl 2001;15(3):200–22.
[10] Good J, Bright J. An object-oriented software framework for the farm-scale simulation of nitrate leaching from agricultural land uses–IRAP FarmSim. In: International congress on modelling and simulation. Australia and New Zealand: Modelling and simulation society; 2005.
[11] Henderson B, Gerber P, Hilinski T, Falcucci A, Ojima D, Salvatore M, Conant R. Greenhouse gas mitigation potential of the world's grazing lands: modeling soil carbon and nitrogen fluxes of mitigation practices. Agric Ecosyst Environ 2015;207:91–100.
[12] Hillyer C, Bolte J, van Evert F, Lamaker A. The modcom modular simulation system. Eur J Agron 2003;18(3–4):333–43.
[13] Keating B, Carberry P, Hammer G, Probert M, Robertson M, Holzworth D, Huth N, Hargreaves J, Meinke H, Hochman Z, et al. An overview of apsim, a model designed for farming systems simulation. Eur J Agron 2003;18(3):267–88.
[14] Machado C, Morris S, Hodgson J, Arroqui M, Mangudo P. A web-based model for simulating whole-farm beef cattle systems. Comput Electron Agric 2010;74(1):129–36.
[15] Martin G, Magne MA. Agricultural diversity to increase adaptive capacity and reduce vulnerability of livestock systems against weather variability – a farm-scale simulation study. Agric Ecosyst Environ 2015;199:301–11.
[16] Mateos C, Zunino A, Campo M. A survey on approaches to gridification. Software: Pract Exp 2008;38(5):523–56.
[17] Moore A, Eckard R, Thorburn P, Grace P, Wang E, Chen D. Mathematical modeling for improved greenhouse gas balances, agro-ecosystems, and policy development: lessons from the Australian experience. Wiley Interdisciplinary Rev: Clim Change 2014;5(6):735–52.
[18] Moreno-Vozmediano R, Montero RS, Llorente IM. Key challenges in cloud computing: enabling the future internet of services. IEEE Internet Comput 2013;17(4):18–25.
[19] Pannell D. On the estimation of on-farm benefits of agricultural research. Agric Syst 1999;61:123–34.

[20] Romera A, Morris S, Hodgson J, Stirling D, Woodward S. A model for simulating rule-based management of cow-calf systems. Comput Electron Agric 2004;42(2):67–86.
[21] Sherlock R, Bright K. An object-oriented framework for farm system simulation. In: MODSIM99-international conference on modelling and simulation. Modelling and Simulation Society of Australia and New Zealand; 1999. p. 783–8.
[22] Corbellini A, Mateos C, Zunino A, Godoy D, Schiaffino S. Persisting big data: the NoSQL landscape. Inf Syst 2016;63:1–23.
[23] Tanenbaum A. Modern operating systems. Pearson Education; 2009.
[24] Tang M, Lee BS, Tang X, Yeo CK. The impact of data replication on job scheduling performance in the data grid. Future Gener Comput Syst 2006;22(3):254–68.
[25] Thain D, Tannenbaum T, Livny M. Distributed computing in practice: the condor experience. Concurrency Comput: Pract Exp 2005;17(2–4):323–56.
[26] Zhao G, Bryan B, King D, Luo Z, Wang E, Bende-Michl U, Song X, Yu Q. Large-scale, high-resolution agricultural systems modeling using a hybrid approach combining grid computing and parallel processing. Environ Modell Software 2013;41:231–8.