КОМПЬЮТЕРНЫЕ ИССЛЕДОВАНИЯ
И МОДЕЛИРОВАНИЕ 2015 Т. 7 № 3 С. 565−571

Ки&М

СЕКЦИОННЫЕ ДОКЛАДЫ

УДК: 004.75

Defining volunteer computing: a formal approach

A. Cs. Marosi[a], R. Lovas[b]

Institute for Computer Science and Control, Hungarian Academy of Sciences,
1518 Budapest, P.O.Box. 63., Hungary

E-mail: [a]marosi.attila@sztaki.mta.hu, [b]lovas.robert@sztaki.mta.hu

Received January 28, 2015

Volunteer computing resembles private desktop grids whereas desktop grids are not fully equivalent to volunteer computing. There are several attempts to distinguish and categorize them using informal and formal methods. However, most formal approaches model a particular middleware and do not focus on the general notion of volunteer or desktop grid computing. This work makes an attempt to formalize their characteristics and relationship. To this end formal modeling is applied that tries to grasp the semantic of their functionalities — as opposed to comparisons based on properties, features, etc. We apply this modeling method to formalize the Berkeley Open Infrastructure for Network Computing (BOINC) [Anderson D. P., 2004] volunteer computing system.

Keywords: BOINC, ASM, Formalism, Volunteer Computing

Определение добровольных вычислений: формальный подход

А. К. Мароши, Р. Ловаш

Институт информатики и управления, Венгерская Академия Наук (MTA SZTAKI),
Венгрия, 1518, г. Будапешт, почтовый офис. 63.

Добровольные вычисления напоминают частные desktop гриды, тогда как desktop гриды не полностью эквивалентны добровольным вычислениям. Известны несколько попыток отличить и категоризировать их, используя как неофициальные, так и формальные методы. Однако, наиболее формальные подходы моделируют специфическое промежуточное ПО (middleware) и не сосредотачиваются на общем понятии добровольного или desktop грид. Эта работа и есть попытка формализовать их характеристики и отношения. Для этой цели применяется формальное моделирование, которое пытается охватить семантику их функциональных возможностей — в противоположность сравнениям, основанным на свойствах, особенностях, и т. п. Мы применяем этот метод моделирования с целью формализовать добровольную вычислительную систему Открытой Инфраструктуры Беркли для сетевых вычислений (BOINC) [Anderson D. P., 2004].

Ключевые слова: BOINC, ASM, Формализм, Добровольные Вычисления

The research presented in this paper was supported by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no 312297 (IDGF-SP).

Citation: *Computer Research and Modeling*, 2015, vol. 7, no. 3, pp. 565–571.

1. Introduction

Desktop Grids (DGs) and Volunteer Computing (VC) utilize the idle computing cycles of desktop computers to solve embarrassingly parallel type of compute-intensive problems, such as Monte Carlo simulations or master-worker type applications. Publicly operated ones using mostly volunteer resources are referred as volunteer computing, or recently as "crowd computing". Contrary, private desktop grids are operated within an organization (i. e., university or company) using the computing resources and applying their local policies. There are several attempts to distinguish and categorize DGs and VC using informal and formal methods [Characterizing and Classifying Desktop Grid, 2007; A Taxonomy of Desktop Grids..., 2008; Wang Y., He H., & Wang Z., 2009]. However, most formal approaches model a particular middleware and do not focus on the general notion of volunteer or desktop grid computing. In this work formal modeling is applied that tries to grasp the semantic of their functionalities — as opposed to comparisons based on properties, features, etc. The result of this work is a formal model of BOINC that aims at serving as a foundation for formalizing other volunteer computing systems and helps categorizing existing middleware. The model is developed using the abstract state machines (ASMs) framework and builds on a model that formalized (service) Grid Computing in general. The paper is organized as follows: the next section summarizes the Abstract State Machines framework. Section 3 discusses related work. Section 4 details the formal model for BOINC, and finally section 5 concludes the paper.

2. Abstract State Machines

The Abstract State Machine is a mathematically well-founded framework for high-level system design and analysis [Borger E. & Stark R. F., 2003] originally introduced as evolving algebras by Gurevich [Gurevich Y., 1993]. ASM allows hiding easily the non-important details at the high-level design phase by formulating the model on a conceptional level rather than based on implementation details and attributes. Lower detail characteristics can be added to the models later gradually. It is an agent based modeling system where the system is described from the perspective of an agent. In ASM states are represented as algebras, i. e., basic sets (called universes) with functions and relations interpreted on them. A signature (or vocabulary) is a finite set of function names each with fixed arity. It also contains the usual Boolean operators (e.g., $\wedge,\vee$) and the symbols true, false, = and undef. A state S of signature Υ is a nonempty set X together with interpretations of function names in Υ on X. X is called the super universe. A nullary function name is interpreted as an element of X this corresponds to the notion of variables. An r-ary function name is interpreted as a function from $X^r \to X$. A location of S is a pair l=(f,x), where f is a function name of arity r in vocabulary Υ and x is an r-tuple of elements of X. The element $f(x)$ is the content of location l. An update is a pair z=(l,y), where l is a location and y is another element of X. Firing x at state S means putting y into the location l while other locations remain unchanged. The resulting state is S' (the sequel of S), thus the interpretation of a function f at argument x has been modified producing a algebra, i. e., a new state. The special nullary Self-function is used to represent the agent and also allows to identify itself amongst other agents. Different agents interpret it differently. This Self-function can never be the subject of updates. ASM models are defined as a set of transition rules.

3. Related work

Choi et al. [Characterizing and Classifying Desktop Grid, 2007; A Taxonomy of Desktop Grids..., 2008] state that DGs have received increased attention for executing high throughput workloads as resources are becoming less expensive. They argue that DGs are different from service grids in many aspects, but there is no taxonomy or survey on DGs. They categorize DGs based on organiza-

tion (centralized or distributed), platform, scale (Internet or LAN) and resource providers (volunteer or enterprise) characteristics. They also compare VC (they refer it as volunteer desktop grids) and DGs (referred as enterprise desktop grids by the authors) to service grids on an informal per attribute basis, and provide no insight what the relation between the DGs and service grids could be. Wang et al. [Wang Y., He H., & Wang Z., 2009] uses a formal method inspired by Mobile Ambients to build a formal model for VC by identifying the different roles for hosts in VC and describing their relation and interaction. The model is derived mainly based on the characteristics of XtremWeb(-HEP) [Computing on large-scale distributed systems..., 2005]. They state that their model can help to lay a strong foundation for further research on formalisms of VC. However their model does not distinguish between DGs and VC and seems generic in an extent that most DG systems could fit it as well. Also it seems their generic model is derived from a single specific middleware: XtremWeb-HEP (XWHEP). They do not validate their assumption that XWHEP indeed is a volunteer computing middleware. For example in [Characterizing and Classifying Desktop Grid, 2007] Choi et al. state: "Lack of trust: In Desktop Grid, anonymous nodes can participate as a resource provider. Some malicious resource providers tamper with the computation and then return corrupted results. *A scheduler should guarantee the correctness of results". In their comparison of volunteer and enterprise DGs result certification is listed for VC. However based on the documentation* [XtremWeb-HEP documentatio, 2014] XWHEP does not provide this functionality: "Result certification: The XWHEP middleware does not propose anything on this field. It is the end user responsibility to verify the results of her jobs". This contradicts the assumptions for the model by Wang et al. [Wang Y., He H., & Wang Z., 2009].

A formal model for (service) grids based on ASM was presented by Nemeth et al. in [Németh Z., & Sunderam V., 2003] and was refined later by Kertesz et al. [Kertész A., & Németh Z., 2009]. Originally Nemeth et al. compared Grids with other distributed systems based on operational differences. They proposed a definition for Grids based on (runtime) semantics of the systems rather than comparing their static characteristics.

In their ASM model Nemeth et al. consider an application (members of universe APPLICATION) as consisting of several processes (universe PROCESS). All processes are owned by a user (USER) and need resources to perform work. Abstract resources are present in resource request and are represented by the ARESOURCE universe, while the PRESOURCE universe represents physical resources allocated to processes. Processes execute a specific task (universe TASK). The physical representation of a task is a static realization of a running process, thus it must be present on the same node (universe NODE) where the process is. This is represented by the *installed: TASK* × *NODE*→ {*true, false*} relation. Nodes, tasks and resources have certain attributes (universe ATTR). A subset of ATTR is the architecture type represented by universe ARCH. The relation *compatible: ATTR* × *ATTR* → {*true, false*} denotes whether to attributes are compatible according to some reasonable definition. A user can login to certain nodes if *CanLogin: USER* × *NODE*→ {*true, false*} evaluates to true. A user is authorized to use given resource if the *CanUse: USER* × *PRESOURCE*→ *{true, false}* relation evaluates to true. The model is centered on processes and their life cycle is described by their states using the state *PROCESS* → *{running, waiting, receive_waiting}* function. In grids the resource requests can be satisfied from various nodes in various ways. The user and the application has no information about the state of the pool of resources a new agent executing module is needed that handles the mapping between them, thus the *resource mapping* functionality is introduced that provides the mapping via the *PROCESS* × *ARESOURCE* → *PRESOURCE* function. It does not specify how resources are actually chosen (it is rather an implementation detail), only assures that compatible physical resources are mapped to each resource request using the compatible: *ATTR* × *ATTR* → *{true, false}* relation. In grids the fact that a user can access the pool of resources does not mean that she can login to the nodes providing the resources $\forall u \in USER, \forall r \in PRESOURCE$, $\forall n \in NODE : CanUse(u,r) \not\Rightarrow CanLogin(u,n)$. Resources are granted by the operating system to processes on the same node, thus a process of the application — belonging to the user — must be present on the node. However users are not authorized to login and start processes. This contradiction is resolved by providing a mapping between the real person, the user who has credentials to access to the

resources of the pool (*globaluser*) and the user — not necessarily a real person — who has login rights on the node (*localuser*). The *user mapping* functionality provides this mapping. The model is a distributed multi-agent ASM where the agents are processes, i. e., elements of the PROCESS universe. It is depicted from the perspective of the processes, where the *Self*-function is represented as $p \in PROCESS$, i. e., different agents interpret p differently.

4. A formal model for BOINC

The here presented model for BOINC is part of a series of models introduced in [Marosi A. Cs., & Nemeth Z., 2013]. It builds on a previous model for volunteer computing (VC) in general, which is shown as MVC-VOTE on Fig. 1/c. The firs model in the series ($M_{GROUND\text{-}DG}$) is based on the model presented in [Németh Z., & Sunderam V., 2003; Kertész A., & Németh Z., 2009]. In this paper only the BOINC model (M_{BOINC}) is discussed, but where necessary details from the previous models are included. The model presented here is a multi-agent ASM model where agents are jobs (i. e., elements from the JOB universe). The nullary *self* function $j \in JOB$ allows an agent to identify itself among other agents. The different agents interpret it differently. Its rules form a module, i. e., a single-agent model that is executed by each agent. Due to space constraints the model transition rules — including the initial state — are going to be detailed in a future paper.

BOINC is a widely used volunteer computing framework with more than 70 public deployments around the world. A deployment of BOINC is generally referred as a project. The formal model presented here aims to capture the major semantics of BOINC as follows: (i) BOINC follows centralized client-server architecture. (ii) Applications cannot be submitted as part of jobs. Jobs must refer a previously at the project deployed application. An administrator must register applications by hand at the project application repository. (iii) BOINC implements a result certification mechanism based on comparing returned finished job instances. Result validators must be supplied on a per application basis. It is an application specific task to determine whether to job instances can be considered as matching or not. (iv) Its owner based on a per application basis can restrict access to a host. Each donor is able to filter the applications of the different BOINC project she contributes to. Finally (v) as reward and incentive donors are awarded virtual credits for each job instance they successfully complete based on the amount of contributed processor time.

Fig. 1/a summarizes the universes for the model only the new and changed components compared to [Németh Z., & Sunderam V., 2003; Kertész A., & Németh Z., 2009] are discussed here. The PLATFORM universe is introduced to represent the different preset combination of operating system and system architecture requirements of applications in BOINC. The REPOSITORY universe represents all application repositories where applications are deployed. Other new universes are as follows. USER: A user of BOINC is the entity that submits jobs and retrieves results. There might be multiple users each responsible for their own jobs. NODE: Umbrella term for user interfaces, managers and hosts (see below). UI: user interface (UI). A node type from where users can submit jobs to BOINC. It acts as a gateway and the pool of resources can be accessed through it. MANAGER: The main component of a desktop grid. It is a type of node that manages resources and allocates Jobs to hosts. Jobs are submitted through UIs to managers. HOST: provides physical resources (PRESOURCE) for jobs and thus, executes them. Hosts are computers of a lab, office and etc. resources for BOINC. The worker component is installed on the hosts. This worker acts as a handler on behalf of the host for BOINC. Since all hosts have workers installed these won't be distinguished in the model rather, only the host referenced with different context. TASK: The physical representation of a job installed on a host. All processes of a specific job executing on a host are represented by a task. A unit of work represented by the UNITOFWORK universe incorporates all data and metadata that can be specific for a job (e.g., command line parameters or input data and required libraries). The DONOR universe is introduced to represent the owners of the hosts who donate theirs to the BOINC project (i. e., *the volunteers*). Fig. 1/b maps the universes of the model to their counterparts in BOINC. Based on the identified major semantics the following functionalities compose the model:

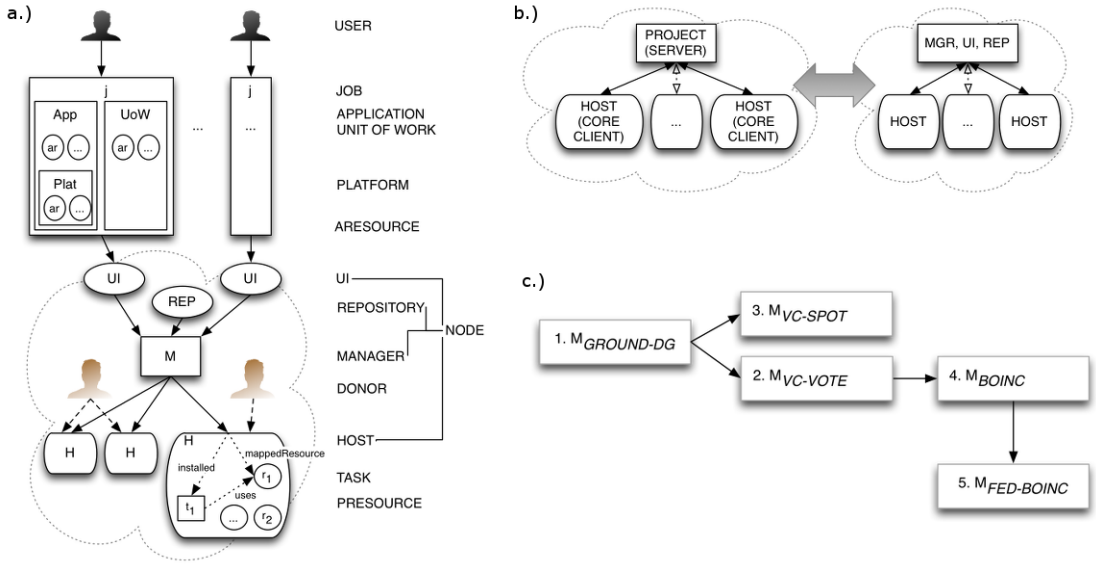

Fig. 1 a.: universes used in the model, b.: the components used in the model and their counterparts in BOINC, and c.: the place of the BOINC model in the series of formal models for DGs and VC

(1) *Resource mapping*. Resource requests of applications are consolidated into platforms in BOINC. During resource mapping it must be ensured that the host supports one of the platforms the application has available. Applications can have multiple implementations, each for a different platform, thus the selected platform is rather mapped to the job instead of the application. Still *unitofworks* have resource requests and mapped resources unchanged. The *supportsPlatform: APPLICATION* × *PLATFORM* → *{true, false}* function tells if an application has an instance for the given platform. This must platform must match the platform the host is reporting. The mapped platform of the job is represented by the mappedplatform: *JOB* → *PLATFORM* function. The platfotm: *{HOST, JOB}* → *PLATFORM* returns a platform the entity supports (there is no restriction that only a single one can be supported). The donor has the privilege to select allowed applications for her hosts as she wishes. BOINC projects usually run a single application aimed at solving some (grand) scientific challenge. However there are *umbrella* projects that host different applications. In this case the donor is given the freedom to disable application that she does not wish to support by accepting their jobs. This is represented by the *appAllowed: DONOR* × *APPLICATION* → *{true, false}* relation.

(2) *User abstraction*. BOINC provides a client program — the worker — that is installed every host and provides the *global user* to *local user* mapping. The worker acts on behalf of the host as a handler for the physical resources and provides the local part for the user mapping. In BOINC the *globaluser* → *localuser* mapping is straightforward since all users of BOINC have access to all hosts. The only restriction is imposed by the allowed applications list of donors.

(3) *Delayed jobs*. BOINC relies on non-dedicated, volatile resources, thus the possibility when a job is delayed for any reason (or even lost) must be taken in account. The cause for such delay can be numerous, e.g., resource requirements cannot be satisfied, or the host the job is mapped is claimed by its owner or is permanently shut down. This is represented by the high-level *jobDekayed: JOB*→ *{true, false}* monitored function. In BOINC a deadline is set for each job (result) to finish. Once the deadline passes the job is overdue and it is aborted by sending the ABORT event to the job: event (*j*):=ABORT.

(4) *Mapped resources become unavailable*. Owners of the resources are prioritized in BOINC and thus resources mapped and assigned to jobs can get unavailable for shorter or longer periods of time (i. e., the owner claims her computer). Here the presource: *JOB* → *PRESOURCE* function denotes the mapped physical resources of the job. The $pr \in PROCESS, h \in HOST : provides(pr, h) = false$ function denotes that the host h (through the worker software) can not provide (some or all of) the mapped

physical resources for the job any more. In this case the job must be suspended using the suspend event. The actual revocation (*users* (*j, presource (j*)):=*false*) happens in the state transition rule for suspend. It is assumed that the unavailability is a temporal state and not all of the resources need to be released, thus the job only needs to be suspended while so. If the resources are available again (represented by provides (*pr,h*)) = *true*) and the job is not using the resources (represented by *users* (*j, pr*):=*false*) task can be started.

(5) *Donor interaction*. Donors have direct control over the host(s) they donate to BOINC. They can suspend, resume computation in general and force to start, stop and abort specific tasks on their hosts. These are represented as events in the model. Specific events can be sent to jobs and the transition rules interpret these. Also *the maskEvents: JOB* $\times$ 2^{EVENT} $\times$ *MASKENTITY* $\rightarrow$ *{true, false}* relation allows to mask specific events from a given job. This allows e.g., if a user suspended all computation on her host, then no task will receive the start event, thus none is allowed to start. Not just the donor can generate events, but also the host, the MASKENTITY tuple represents these two entities (MASKENTITY ={donor, host}).

(6) *Result validation*. Comparing returned results is an application specific task in BOINC since it depends on the application which results can be considered matching and which not. The validator components provide this functionality. There must be a validator provided for each application. The comparison of results is performed by the checkset and checkpair functions. Each of the validator modules provides an implementation for these functions that check the successful results of a single application. BOINC also provides validators for some common cases that can be reused by applications, e.g., a bitwise validator that compares results bit-by-bit. The functions are used by the doValidation macro that provides the common functionality for all validator modules. Ultimately the module determines if the successful finished results indeed produced correct outputs. This state is represented by the validateState function of the results: valid, invalid or inconclusive if no decision could be made. Validation is achieved in two ways depending on if a representative result ("canonical" result denoted by the *canonicalresult: JOB* $\rightarrow$ *JOB function*) is already found for the work unit. If it was already found (*canonicalResult(j)* $\neq$ *undef*) then all new results are compared against it (using the checkPair function). The outcome of this comparison can be that either the results match thus the new one is valid (*validateState(r) = valid*) or they mismatch and the result is invalid. If there is no canonical result available yet then first it is checked whether there are enough successful results available to form a quorum and select one (the number is determined by the minQuorum function). Next a check is run on the set of results with checkSet. This function compares all results, decides whether they are valid or invalid and selects a canonical result from the valid ones. It is still a possibility that no canonical result is found (*validateState(r) = inconclusive* for all results). In this case the validation procedure is rerun later when a new successful result is returned. However if the limit for successful results is reached and still there is no consensus on the validation the work unit is considered failed. If there is no consensus but the limit is not reached then the targetNResults is increased for the work unit and a new result (job instance) will be created. If a canonical result is found by checkset then there is no need to send the unsent results to clients, thus abort event is generated for them. However already in progress results should be accepted (and validated) when they are returned. For valid results the validator grants credit (represented by the grantCredit function).

The here presented functionalities (with the transition rules and initial state) form a formal model of BOINC.

5. Conclusions

In this paper a formal model for BOINC was presented using the ASM method. The model (see M_{BOINC} on Fig. 1/c) is based on a series of models for Desktop Grid and Volunteer Computing [Marosi A. Cs., & Nemeth Z., 2013] and a formal model for Service Grids defined in [Németh Z., & Sunderam V., 2003; Kertész A., & Németh Z., 2009]. This model has three goals. First a validation for the previous models in the series: a real VC system can be modeled using them. Second it aims to be

a foundation for formalizing other volunteer computing systems. Finally the model acts as a basis for the next model in the series (see $M_{FED\text{-}BOINC}$ on Fig. 1/c), which models a novel method for federating distinct volunteer computing projects and enables workload sharing.

References

Anderson D. P. BOINC: A System for Public-Resource Computing and Storage. In R. Buyya (Ed.) // Fifth IEEE/ACM International Workshop on Grid Computing. — 2004. — P. 4–10.

Borger E. & Stark R..F. Abstract State Machines: A Method for High-Level System Design and Analysis. Secaucus, NJ, USA: Springer-Verlag New York, Inc. 2003.

Cappello F., Djilali S., Fedak G., Herault T., Magniette F., Neri V. & Lodygensky O. Computing on large-scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid // Future Generation Computer Systems. — 2005. — 21(3). — P. 417–437.

Choi S., Buyya R., Kim H., Byun E. & Baik M. A Taxonomy of Desktop Grids and its Mapping to State-of-the-Art Systems // ACM Computing Surveys. — 2008. — V. 1–61.

Choi S., Kim H., Byun E., Baik M., Kim S., Park C., & Hwang C. Characterizing and Classifying Desktop Grid // Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07). — 2007. — P. 743–748 (doi:10.1109/CCGRID.2007.31).

Gurevich Y. Evolving algebras: An attempt to discover semantics // Current Trends in Theoretical Computer Science. — 1993. — P. 1–27.

Kertész A., & Németh Z. Formal Aspects of Grid Brokering // Electronic Proceedings in Theoretical Computer Scienceю — 2009. — 14. — P. 18–31 (doi:10.4204/EPTCS.14.2).

Marosi A. Cs., & Nemeth Z. Two Sides of a Coin: Formalizing Volunteer and Desktop Grid Computing. In M. Bubak, M. Turała, & K. Wiatr (Eds.) // Proceedings of the Cracow Grid Workshop. — 2013. — P. 69–70. Kraków: ACK CYFRONET AGH.

Németh Z., & Sunderam V. Characterizing grids: Attributes, definitions, and formalisms // Journal of Grid Computing. — 2003. — P. 9–23.

Wang Y. He, H, & Wang Z. Towards a formal model of volunteer computing systems // In 2009 IEEE International Symposium on Parallel & Distributed Processing (p. 1–5). IEEE. — 2009 (doi:10.1109/IPDPS.2009.5161137).

XtremWeb-HEP documentation. http://www.xtremweb-hep.org/lal/doc/xwhep-intro-1.4.0.pdf, P. 30. Last accessed on 2014-02-01.