Inna Pereverzeva | Michael Butler | Asieh Salehi Fathabadi | Linas Laibinis | Elena Troubitsyna

Formal Derivation of Distributed MapReduce

Turku Centre *for* Computer Science

TUCS Technical Report
No 1099, January 2014

Formal Derivation of Distributed MapReduce

Inna Pereverzeva
Åbo Akademi University, Department of Computer Science
Turku Centre for Computer Science
inna.pereverzeva@abo.fi
Michael Butler
University of Southampton, UK
mjb@ecs.soton.ac.uk
Asieh Salehi Fathabadi
University of Southampton, UK
asf08r@ecs.soton.ac.uk
Linas Laibinis
Åbo Akademi University, Department of Computer Science
linas.laibinis@abo.fi
Elena Troubitsyna
Åbo Akademi University, Department of Computer Science
elena.troubitsyna@abo.fi

TUCS Technical Report
No 1099, January 2014

Abstract

MapReduce is a powerful distributed data processing model that is currently adopted in a wide range of domains to efficiently handle large volumes of data, i.e., cope with the big data surge. In this paper, we propose an approach to formal derivation of the MapReduce framework. Our approach relies on stepwise refinement in Event-B and, in particular, the *event refinement structure* approach – a diagrammatic notation facilitating formal development. Our approach allows us to derive the system architecture in a systematic and well-structured way. The main principle of MapReduce is to parallelise processing of data by first mapping them to multiple processing nodes and then merging the results. To facilitate this, we formally define interdependencies between the map and reduce stages of MapReduce. This formalisation allows us to propose an alternative architectural solution that weakens blocking between the stages and, as a result, achieves a higher degree of parallelisation of MapReduce computations.

Keywords: formal modelling, Event-B, refinement, event refinement structure, MapReduce.

TUCS Laboratory
Embedded Systems Laboratory (ESLAB)

1 Introduction

MapReduce is a widely used framework for handling large volumes of data [5]. It allows the users to automatically parallelise computations and execute them on large clusters of computers. Essentially, the computation is performed in two stages – map and reduce. The first stage maps the input data to multiple processing nodes, while the second stage performs parallel computations to merge the obtained results. Typically, execution of the map stage is blocking, i.e., execution of the reduce stage does not start until the map stage is completed. Though MapReduce is already a highly performant framework, to keep pace with the drastically increasing volume of data, it would be desirable to loosen the coupling between the stages and hence exploit the potential for parallelisation to the fullest.

In this paper, we undertake a formal study of the MapReduce framework. We formally model the control flow and data interdependencies between the map and reduce tasks, as well as derive the conditions under which the execution of the reduce stage can overlap with the execution of the map stage. Our formalisation of the (generic) MapReduce framework relies on the Event-B method and the associated Rodin platform. Event-B [1] is a formal approach that is particularly suitable for the development of distributed systems. The system development in Event-B starts from an abstract specification that is transformed into a detailed specification in a number of correctness-preserving refinement steps. In this paper, the Event Refinement Structure approach [3, 6] is used to facilitate the refinement process. The technique provides us with an explicit graphical representation of the relationships between the events at different levels of abstraction and helps to gradually derive the complex MapReduce architecture.

Event-B relies on proof-based verification that is integrated into the development process. The Rodin platform [10] automates development in Event-B by generating the required proof obligations and automatically discharging a significant part of them. Via abstraction, proof and decomposition, Event-B enables reasoning about system-level properties of complex distributed systems. In particular, it allows us to explicitly define interdependencies between the processed data and derive the conditions under which an execution of the reduce stage can start before completion of the map stage. We believe that the proposed approach provides the designers with a formally grounded insight on the properties of MapReduce and enables fine-tuning of the framework to achieve a higher degree of parallelisation.

The rest of the paper is organised as follows. In Section 2 we describe the generic MapReduce framework and our formalisation of it. In Section 3 we give an overview of the Event-B formalism and the Event Refinement Structure (ERS) approach. In Section 4 we present our formal derivation of the MapReduce framework in Event-B using the ERS approach. As a

result, we derive two alternative architectures of the MapReduce framework – blocking and partially blocking. Finally, in Section 5 we overview the related work and present some concluding remarks.

2 MapReduce

2.1 Overview of MapReduce

In this section we give an overview of the MapReduce framework. MapReduce is a programming model for processing large data sets. It has been originally proposed by Google [5]. The framework is designed to orchestrate the work on distributed nodes, run various computational tasks in parallel, providing at the same time for redundancy and fault tolerance. Distributed and parallelised computations are the key mechanisms that make the MapReduce framework very attractive to use in a wide range of application areas: data mining, bioinformatics, business intelligence, etc. Nowadays it is becoming increasingly popular in cloud computing. There exist different implementations of MapReduce, among them open-source Hadoop [2], Hive [11], Pig [7], and others.

The MapReduce computational model was inspired by the *map* and *reduce* functions widely used in functional programming. A MapReduce computation is composed of two main steps: the *map stage* and the *reduce stage*. During the map stage, the system inputs are divided into smaller computational tasks, which are then performed in parallel (provided there are enough processors in the cluster). The obtained collective results then become the inputs for the reduce stage, which combines them in some way to produce the overall output. Once again, the reduce inputs are split into smaller computational tasks that can be executed in parallel.

The MapReduce framework can be tuned to perform different data transformations by the user-supplied map and reduce functions. These functions encode basic mapping and reduction tasks to be performed in single nodes. The MapReduce framework then incorporates the provided functions and orchestrates the overall distributed computations based on them.

A typical example illustrating MapReduce computations is counting the word occurrences in a large set of documents. The input data set is split into smaller portions and the user-provided map function is applied to each such data block. The *map* function simply assigns to each word it encounters the value equal to 1. Overall, the map stage produces a collection of (word,1) pairs as intermediate results. Then, during the reduce stage, the user-supplied reduce function takes a portion of these intermediate data related to a particular word and sums all the occurrences of that word. Such a computation is done for each encountered word. The overall result is a set of (word,number) pairs.

2.2 Towards Formal Reasoning about MapReduce

In this section, we present a formalisation of the MapReduce framework. Specifically, we mathematically represent all MapReduce execution stages, i.e., the required data and control flow, and identify the computational (map and reduce) tasks that can be executed in parallel. Moreover, we formally define possible data interdependencies between the map and reduce tasks. The latter allows us to propose an alternative architectural solution, which weakens blocking between the MapReduce phases and, as a result, achieves a higher degree of parallelisation of MapReduce computations. In Section 4, we will propose two alternative formal developments of the MapReduce framework in Event-B, both of which rely on the formalisation presented below.

Let *IData* be an abstract type defining the input data to be processed within the MapReduce framework and *OData* be an abstract type defining the resulting output data. In a nutshell, a MapReduce computation processes the given input data and generates some result. Thus, it can be formally represented as a function:

$$MapReduce \in IData \rightarrow OData.$$

More specifically, a MapReduce computation can be defined as a functional composition of the following phases: *MSplit*, *Map*, *RSplit*, *Reduce*, and *Combine*:

$$MapReduce = MSplit;\ Map;\ RSplit;\ Reduce;\ Combine.$$

The MapReduce process starts with the *MSplit* phase. During this phase, the input data are split into a number of blocks (portions of the input data), which can be handled independently of each other. In the following *Map* phase, the user-provided *map* function is applied to each such input block. Next, in the *RSplit* phase, the MapReduce framework groups together all the intermediate results obtained after the *Map* phase to prepare for the reduce computations. Similarly to the *MSplit* phase, the data are divided into blocks that can be handled separately. After that, the *Reduce* phase is executed, during which the user-supplied *reduce* function is repeatedly applied (once per each block). Finally, in the *Combine* phase, all the obtained results are combined into the final output. Let us note that in some MapReduce implementations the last *Combine* phase might be omitted.

Formalisation of the MapReduce execution phases. Next we define all the MapReduce execution phases in more detail. In the *MSplit* phase, the input data are split into a number of blocks that are later supplied to the *map* function. To emphasise the independent nature of map computations, we associate the notion of a *map task* with such a portion of the input data to be processed separately.

Let *MTask* be a set of all possible map tasks and *MData* be an abstract type defining the data obtained after the splitting. Then the *MSplit* phase can be mathematically represented as follows:

$$MSplit \in IData \rightarrow (MTask \nrightarrow MData).$$

Essentially, *MSplit* produces a partitioning of the input data to be used in the *Map* phase among different map tasks. Note that the result of *MSplit* is a partial function since only a subset of *MTask* may be needed for particular input data.

We assume that the input data fully determines the number and the subset of involved map tasks.[1] To extract this information, we use the following functions

$$mtasks \in IData \rightarrow \mathbb{P}_1(MTask), \quad mnum \in IData \rightarrow \mathbb{N}_1$$

defined as

$$\begin{aligned} \forall idata \in IData \cdot\ mtasks(idata) &= \mathsf{dom}(MSplit(idata)), \\ \forall idata \in IData \cdot\ mnum(idata) &= \mathsf{card}(MSplit(idata)), \end{aligned}$$

where dom and card are the function domain and set cardinality operators.

The *Map* phase involves transformation of all the data obtained by the *MSplit* phase into the intermediate form to be used in the later phases. Let *RData* be an abstract type defining the intermediate data obtained after the *Map* phase. Then *Map* phase can be mathematically represented as a following function:

$$Map \in (MTask \nrightarrow MData) \rightarrow \mathbb{P}_1(MTask \times RData).$$

Therefore, *Map* takes the map data partitioning produced by *MSplit* and returns the transformed data associated with the map tasks that produced them. These results then become the input data for the following reduce computations.

In our formalisation the *Map* results consist of a set of (*mtask, rdata*) pairs, without assuming any further structure among them. This is done intentionally, since grouping and partitioning of these data will be performed in the *RSplit* phase.

All the involved map tasks should be performed within the *Map* phase. Formally, this requirement can be formulated as follows:

$$\forall f \in MTask \nrightarrow MData \cdot \mathsf{dom}(f) = \mathsf{dom}(Map(f)).$$

[1] This applies only to the involved computational tasks. Actual software components that will be employed to carry out the necessary computations can be dynamically assigned and re-assigned for a specific map or reduce task.

Next the results obtained by the *Map* phase are grouped together to prepare for reduce computations. Similarly to the *MSplit* phase, they should be first partitioned among the individual *reduce tasks*.

Let *RTask* be a set of all possible reduce tasks. Then the *RSplit* phase can be formally defined as the following function:

$$RSplit \in \mathbb{P}_1(MTask \times RData) \rightarrow (RTask \nrightarrow \mathbb{P}_1(RData)).$$

Essentially, the function takes the intermediate results produced by the *Map* phase and produces data partitioning among the involved reduce tasks.

We can reason about the actual number and the subset of the involved reduce tasks. Once again, this is determined by the original input data. Formally, we introduce the functions

$$rtasks \in IData \rightarrow \mathbb{P}_1(RTask), \quad rnum \in IData \rightarrow \mathbb{N}_1$$

defined as

$$\begin{aligned} &\forall idata \in IData \cdot\ rtasks(idata) \ = \ \mathsf{dom}(RSplit(Map(MSplit(idata)))),\\ &\forall idata \in IData \cdot\ rnum(idata) \ = \ \mathsf{card}(RSplit(Map(MSplit(idata)))). \end{aligned}$$

The *RSplit* phase only rearranges the intermediate data, producing their partitioning among the reduce tasks. Therefore, neither new data should appear nor any of the existing data can disappear during this transformation. Mathematically, this can be formulated as the following property:

$$\forall f \in \mathbb{P}_1(MTask \times RData) \cdot \mathsf{ran}(f) \ = \ (\bigcup rt \in \mathsf{dom}(RSplit(f)) \mid RSplit(f)(rt)),$$

where ran is the function range operator.

The *Reduce* phase is similar to the *Map* phase – it takes as input a data partitioning produced by *RSplit* and returns transformed data:

$$Reduce \in (RTask \nrightarrow \mathbb{P}_1(RData)) \rightarrow \mathbb{P}_1(OData),$$

where *OData* is an abstract type defining the resulting output data.

Finally, the last *Combine* phase can be simply defined as follows:

$$Combine \in \mathbb{P}_1(OData) \rightarrow OData.$$

Formalisation of the `map` and `reduce` functions. The *Map* phase is based on repeated invocations of the user-supplied function `map`. The `map` function can be formally represented in the following way:

$$\texttt{map} \in MData \rightarrow \mathbb{P}_1(RData).$$

Thus, it takes an input data from *MData* and produces some intermediate data to be used in reduce computations. The function `map` and the *Map*

phase are tightly linked. To be precise, the union of all the results obtained from all the `map` function applications should be equal to the overall result of the *Map* phase:

$$\forall idata, f \cdot idata \in IData \land f \subseteq MSplit(idata) \Rightarrow$$
$$Map(f) \;=\; (\bigcup mt \in \mathsf{dom}(f) \mid \{mt\} \times \mathtt{map}(f(mt))).$$

The condition $f \subseteq MSplit(idata)$ essentially requires that this property should be true for any part of the partitioned data. In other words, calculation of any involved map tasks can be carried out completely independently of each other.

The user-supplied `reduce` function can be specified as follows:

$$\mathtt{reduce} \in \mathbb{P}_1(RData) \rightarrow \mathbb{P}_1(OData).$$

It takes as an input a subset of the reduce data *RData* and produces some subset of output data from *OData*.

Finally, the overall result of the *Reduce* phase should be equal to the combined results obtained by repeated application of the `reduce` function:

$$\forall idata, f \cdot idata \in IData \land f \subseteq RSplit(Map(MSplit(idata))) \Rightarrow$$
$$Reduce(f) \;=\; (\bigcup rt \in dom(f) \mid \mathtt{reduce}(f(rt))).$$

Once again, the condition $f \subseteq RSplit(Map(MSplit(idata)))$ means that this property should hold for any subset of reduce tasks.

Formalisation of interdependencies between the map and reduce tasks. The main principle of MapReduce is that all the map and reduce computations are distributed to multiple independent processing nodes. The reduce inputs are based on the previously produced map outputs. However, in some cases, the reduce inputs might depend on only particular map outputs. Therefore, the reduce stage can be initiated before all the map computations are finished. To relax the limitation of the original MapReduce computation flow, requiring that the reduce stage starts only after completing the map stage, we formally define the *dependence relation* between the map and reduce tasks as the following function *dep*:

$$dep \in IData \rightarrow \mathbb{P}(RTask \times MTask),$$

with the following property:

$$\forall\, idata \in IData,\, rt \in RTask,\, mt \in MTask \cdot \;\; rt \mapsto mt \in dep(idata) \;\; \Leftrightarrow$$
$$mt \in \mathsf{dom}(MSplit(idata)) \land$$
$$(\exists rd \in RData \cdot rt \in \mathsf{dom}(RSplit(Map(MSplit(idata))))) \;\land$$
$$rd \in RSplit(Map(MSplit(idata)))(rt) \;\land\; mt \mapsto rd \in Map(MSplit(idata))).$$

The property states that for any input data *input*, a map task *mt* and a reduce task *rt* are in *dependence relation* (i.e., a reduce task depends on

a map task), if and only if some intermediate data rd has been generated for this reduce task rt by the computations of the map task mt during the Map phase. Essentially, the relation dep defines the data interdependencies between the map and reduce stages. This formalisation allows us to propose (in Section 4) an alternative architectural solution that weakens blocking between the stages.

Finally, to make it possible for a particular reduce task to start immediately after all the necessary data have been produced by the map tasks related by dep, we need a version of $RSplit$, defining a partial split related with a specific reduce task. For a given reduce task, it produces the grouped together results obtained within the Map phase:

$$\texttt{rsplit} \in RTask \nrightarrow (\mathbb{P}_1(MTask \times RData) \rightarrow \mathbb{P}_1(RData)).$$

Again, the union of the results obtained from all the `rsplit` function applications should be the result of the $RSplit$ phase:

$$\forall f \in \mathsf{ran}(Map) \cdot RSplit(f) = (\bigcup rt \cdot rt \in \mathsf{dom}(\texttt{rsplit}) \mid \{rt \mapsto \texttt{rsplit}(rt)(f)\}).$$

In Section 4 we will demonstrate that, by relying on the proposed formalisation, we can derive a formal model of the MapReduce framework. There we will propose two models of MapReduce – *blocking* and *partially blocking* models.

3 Formal Development by Refinement: Background

3.1 Event-B

Event-B is a state-based formal approach that promotes the correct-by-construction development paradigm and formal verification by theorem proving [1]. In Event-B, a system model is specified using the notion of an *abstract state machine*. An abstract state machine encapsulates the model state, represented as a collection of variables, and defines operations on the state, i.e., it describes the dynamic behaviour of a modelled system. The variables are strongly typed by the constraining predicates that, together with other important system properties, are defined as model *invariants*. Usually, a machine has an accompanying component, called a *context*, which includes user-defined sets, constants and their properties given as a list of model axioms.

The dynamic behaviour of the system is defined by a collection of atomic *events*. Generally, an event has the following form:

$$e \mathrel{\widehat{=}} \textbf{any } a \textbf{ where } G_e \textbf{ then } R_e \textbf{ end},$$

where e is the event's name, a is the list of local variables, and (the event *guard*) G_e is a predicate over the model state. The body of an event is defined by a *multiple* (possibly nondeterministic) assignment to the system variables. In Event-B, this assignment is semantically defined as the next-state relation R_e. The event guard defines the conditions under which the event is *enabled*, i.e., its body can be executed. If several events are enabled at the same time, any of them can be chosen for execution nondeterministically.

Event-B employs a top-down refinement-based approach to system development. A development starts from an abstract specification that nondeterministically models the most essential functional requirements. In a sequence of refinement steps, we gradually reduce nondeterminism and introduce detailed design decisions. The consistency of Event-B models, i.e., verification of model well-formedness, invariant preservation as well as correctness of refinement steps, is demonstrated by discharging the relevant proof obligations. The Rodin platform [10] provides an automated support for modelling and verification. In particular, it automatically generates the required proof obligations and attempts to discharge them.

3.2 Event Refinement Structure

The Event Refinement Structure (ERS) [3, 6] approach augments Event-B refinement with a graphical notation that allows us to explicitly represent the relationships between the events at different abstraction levels as well as define the required event sequence in a model. ERS is illustrated by example in Figure 1. The diagram explicitly shows that *AbstractEvent* is refined by *Event2*, while *Event1* is a new event that refines *skip*. Moreover, the diagram shows that the effect achieved by *AbstractEvent* in the abstract machine is realised in the refining machine by the occurrence of *Event1* followed by *Event2*.

In ERS, the sequential execution of the leaf events is depicted from left to right. The event sequencing is managed by additional control variables introduced into the underlying Event-B model. For instance, for each leaf event (node) represented in Fig. 1, there is one boolean control variable with the same name as the event. When the event Event1 occurs, the corresponding control variable is set to *TRUE*. The following event, *Event2*, can occur only after *Event1*. This is achieved by checking the value of the *Event1* control variable in the guard of *Event2*.

Boolean variables only allow controlling single execution of events. When multiple executions of an event are needed, the event is parameterised and set control variables are used instead of boolean ones. This allows the event to occur many times with different values of its parameter. A parameter can be introduced in an event by the ERS constructors. The ERS constructors used in this paper are illustrated by two simple examples in Fig. 2. The use

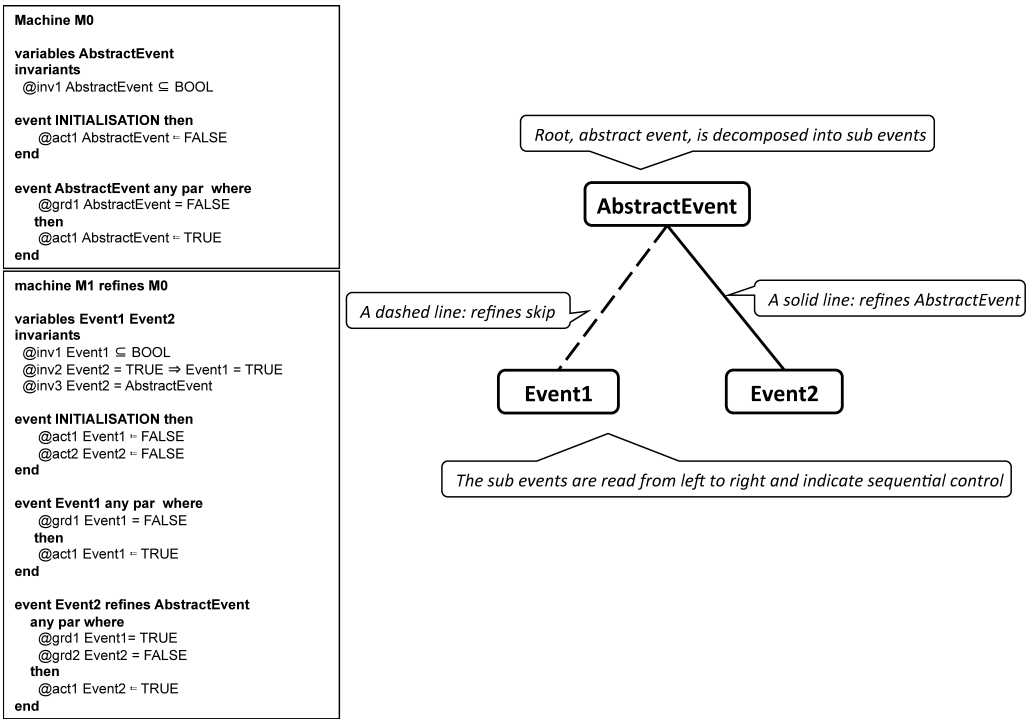


Figure 1: Event Refinement Structure (ERS) Diagram

of *all* constructor indicates that *Event1* is executed for all instances of the p parameter before execution of *Event2*, while the use of the constructor *some* indicates that *Event1* is executed for some of instances of the p parameter before execution of *Event2*. The corresponding control variables for *Event1* and *Event2* are defined as sets in the model.

Figure 2: ERS *all* /*some* Constructors

Event-B adopts an event-based modelling style that facilitate the correct-by-construction development of complex distributed systems. Since MapReduce is a framework designed for large-scale distributed computations, Event-B is a natural choice for its formal modelling and verification.

4 Formal Development with Event Refinement Structure

In this section, we rely on our formalisation presented in Section 2.2 to develop two alternative Event-B models of the MapReduce framework: *blocking*

and *partially blocking*. The presented formal developments make use of the Event Refinement Structure (ERS) approach, presented in Section 3.2. Our development strategy is based on gradually unfolding all the MapReduce computational phases by refinement. Such small model transformation steps allow us to efficiently handle the complexity of the MapReduce framework.

4.1 Blocking Model of MapReduce

The mathematical data structures and their properties from our MapReduce formalisation constitute the basis for defining the Event-B context component that is used throughout the whole formal development. Essentially, the whole presented formalisation is incorporated as the context, e.g.

$\texttt{axm8:}\ MSplit \in IData \rightarrow (MTask \nrightarrow MData)$

$\texttt{axm9:}\ \ Map \in (MTask \nrightarrow MData) \rightarrow \mathbb{P}_1(MTask \times RData)$

$\texttt{axm10:}\ \ RSplit \in \mathbb{P}_1(MTask \times RData) \rightarrow (RTask \nrightarrow \mathbb{P}_1(RData)),\ ...$

We will constantly rely on these definitions to ensure the correctness of the overall data transformation process within our MapReduce models. Since the formalised definitions are still abstract (generic), our presented development essentially formally describes a family of possible MapReduce implementations. Due to space limit, we do not present the complete development but rather give its graphical representation using the ERS graphical notation. The full formal developments can be found in Appendix.

Abstract model of MapReduce. We start with an abstract model in which the whole MapReduce computation is done in one atomic step. This behaviour is modelled by the event OutputMapReduce:

OutputMapReduce $\widehat{=}$
any $t1,\ t2,\ t3,\ t4,\ pmtask,\ prtask$
when $pmtask \subseteq MTask\ \wedge\ prtask \subseteq RTask\ \wedge\ t1 \in pmtask \rightarrow MData\ \wedge$
$t2 \in \mathbb{P}_1(pmtask \times RData) \wedge t3 \in prtask \rightarrow \mathbb{P}_1(RData) \wedge t4 \in \mathbb{P}_1(OData)\ \wedge$
$t1 = MSplit(idata) \wedge t2 = Map(t1) \wedge t3 = RSplit(t2) \wedge t4 = Reduce(t3)$
then $output := Combine(t4)$
end

With help of the ERS approach, we will decompose the atomicity of OutputMapReduce into smaller steps. Verification of the refinement proof obligations ensures that the decomposition preserves correctness. Specifically, in the next several consecutive refinement steps, we break the atomicity of the OutputMapReduce event by introducing explicit events for the following MapReduce phases: *MSplit*, *Map*, *RSplit*, and *Reduce*. Fig. 3 presents the ERS diagram of the model.

The new model events MapSplit, MapPhase, ReduceSplit and ReducePhase specify the sequential execution of the MapReduce phases. The sequence

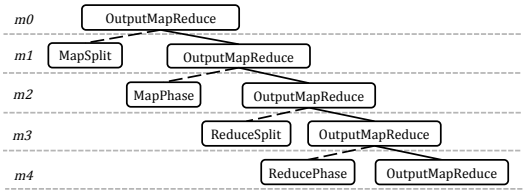


Figure 3: Blocking model: ERS diagram (for OutputMapReduce)

between the events is enforced by following the rules given in Section 3.2. It is also specified by the invariant properties on the control variables:

$$OutputMapReduce = TRUE \Rightarrow ReducePhase = TRUE,$$
$$ReducePhase = TRUE \Rightarrow ReduceSplit = TRUE,$$
$$ReduceSplit = TRUE \Rightarrow MapPhase = TRUE,$$
$$MapPhase = TRUE \Rightarrow MapSplit = TRUE.$$

Moreover, to store the intermediate results of separate phases, we introduce a number of variables ($msplit$, map_result, $rsplit$ and $reduce_result$) that are updated during execution of the corresponding events. The variable updates are also performed according to the formalisation given in Section 2.2. For instance, the variable $msplit$ is introduced to store the result of the *MSplit* phase. After the execution of the MapSplit event, $msplit$ gets the value equal to *MSplit(idata)*.

Breaking atomicity of the Map phase. In the second refinement step, we introduce the event MapPhase that abstractly models the *Map* phase. Essentially, the *Map* phase involves parallel execution of all the map tasks. To introduce such a behaviour, we use the constructor "*all* constructor", which is applied to the MTProcess event that models the execution of a particular map task (see Fig.4). The expression "*all(mt)*" means that the MTProcess event can be enabled for multiple values of $mt \in \mathsf{dom}(msplit)$. On the other hand, the MapCommit event can only occur when all the map computations of map tasks have been finished. In Event-B, we model this by adding a variable $MTProcess$, which is a set containing all possible map tasks that should be processed. The order between the events is ensured by the invariants on the control variables, e.g.,

$$\texttt{inv6:}\ \ MapCommit = TRUE \Rightarrow MTProcess = \mathsf{dom}(msplit),$$

where $\mathsf{dom}(msplit)$ defines the set of all current map tasks. The invariant states that if the MapCommit event has been executed, then all the map tasks have been completed before it. While specifying the MTProcess event, we rely on the definition of the `map` function, given in Section 2.2. Fig.4 shows the ERS diagram for the event MapPhase.

Further refinements of the Map phase. During the MapReduce execution, all the map and reduce tasks are parallelised and distributed to

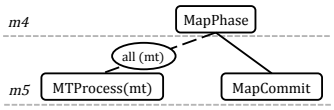


Figure 4: Blocking model: ERS diagram (for MapPhase)

multiple processing nodes – the actual software components that carry out the computations. We name these components as *map* and *reduce workers*. Moreover, there is a special component – *master* – that controls all the computations and assigns the map and reduce tasks to the workers. The master periodically pings every worker. In case of a worker failure, the master reassigns tasks from the failed worker to a healthy one. This procedure can be repeated until the master gets the result for a particular map or reduce task from some worker. To introduce such functionality, we carry out several further refinements focusing on the *Map* phase. These refinements elaborate on modelling of map task execution.

Fig.5 illustrates the event MTProcess and its several consecutive levels of atomicity decomposition. First, the abstract event MTProcess is broken into two concrete events, MTok and MTSuccess correspondingly. The MTok event models the execution of the map task mt by a particular map worker mw. The result of this computation should be approved by the master side, which is modelled by execution of the MTSuccess event. The "*some*" constructor indicates that the event MTok may be executed only for some instances of the mw parameter before the MTSuccess event becomes enabled. The *MTSuccess* and *MTok* control variables are defined as sets, which allows for multiple executions of the MTSuccess and MTok events. Later on, in the next

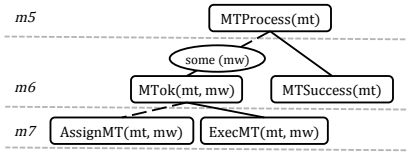


Figure 5: Blocking model: ERS diagram (for MTProcess)

refinement step, the atomicity of the MTok event is broken into two events AssignMT and ExecMT. The event AssignMT models an assignment of a map task mt to a particular map worker mw, while ExecMT models the successful execution of the task by this worker.

Similarly to the *Map* phase, we refine the *Reduce* phase by gradually unfolding its computations. The overall refinement structure is presented on Fig.6.

Let us note that the proposed architecture is *blocking* in the sense that the reduce computations can be only started after all the map computations have been finished. The formal derivation of the blocking model and its dynamics is performed under this condition. Next we propose an alternative

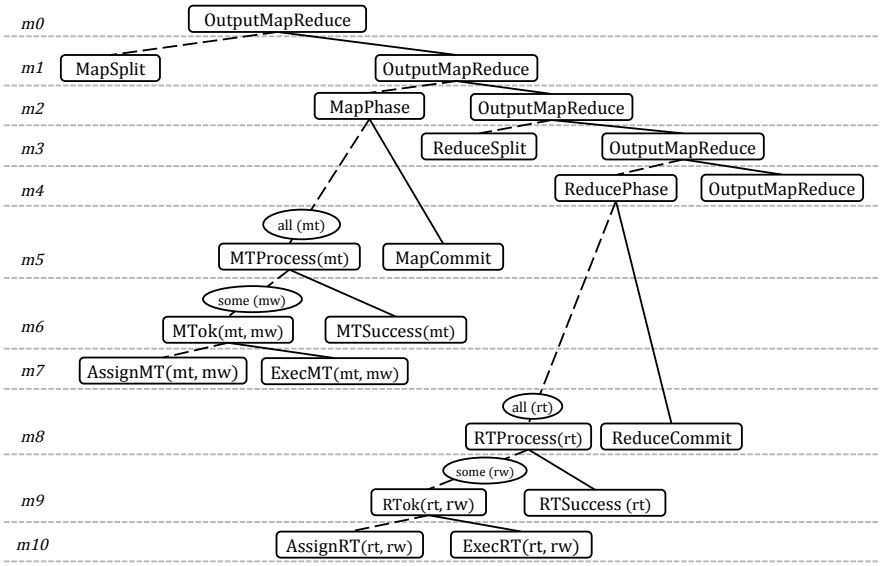
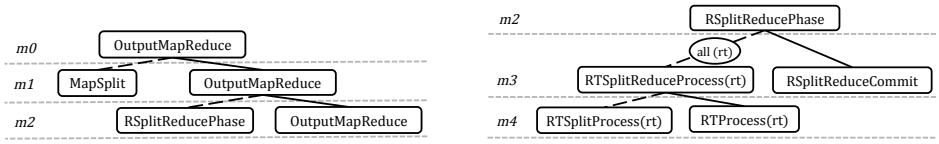


Figure 6: MapReduce ERS Diagram: blocking model

architectural solution of the MapReduce framework that weakens blocking between the map and reduce stages and, as a result, achieves a higher degree of parallelisation of the MapReduce computations. For this purpose, we will make use of the *dependence relation* between map and reduce tasks introduced in the Section 2.2. We call this model *partially blocking model*.

4.2 Partially Blocking Model of MapReduce

We start from the same initial specification as for the blocking model, in which the whole MapReduce computation is done in one atomic step, and then refine it in order to introduce the *MSplit* phase. Next, in contrast to the previous derivation, we separate the phase that combines executions of the *RSplit* and *Reduce* phases – RSplitReducePhase. Fig.7 (a) presents the ERS diagram of the refined model.



(a) (b)

Figure 7: Partially blocking model: ERS diagrams

RSplitReducePhase involves executions of the *RSplit* and *Reduce* phases for all reduce tasks. Essentially, these computations are parallelised. To introduce such behaviour, we use the"*all*" constructor applied to the RTSplitReduceProcess event that, for a particular reduce task rt, performs split and then reduce computations (see. Fig.7 (b)). Next, we separate these split and reduce executions of the particular reduce task rt. Namely, the

event RTSplitReduceProcess is split into two concrete events, RTSplitProcess and RTProcess. Here we again rely on the *rsplit* and *reduce* functions formalised in Section 2.2.

Up to now we did not introduce the Map phase explicitly. However, the results of *MapPhase* are simulated internally, by storing the intermediate results in the local variables of the RTSplitProcess event. To explicitly model the Map phase, the event RTSplitProcess is now split into two events MTProcess and RSplit (see Fig.8). The constructor "all" is parameterised by $(mt \in dep[\{rt\}])$". It means that the event MTProcess is executed for all those map tasks, mt, that are in data dependency with the reduce task rt. Therefore, to start the *RSplit* phase, we do not need to wait until all the map tasks are completed. Here we are relying on the definition of data interdependency *dep* between the map and reduce stages, formalised in Section 2.2. Finally, the MTProcess and RTProcess events are refined in the same manner as in the blocking model presented in the Section 4.1.

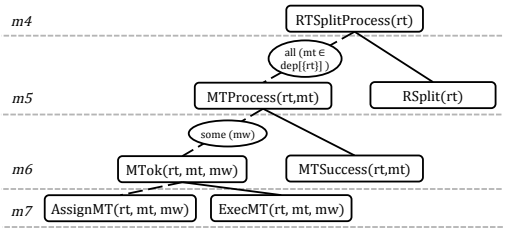


Figure 8: Partially blocking model: ERS diagram (for RTSplitProcess)

Let us note that the proposed partially blocking model allows us to achieve a higher degree of parallelisation of MapReduce computations. Indeed, for a particular reduce task, when the dependent map tasks have already been executed, the *RSplit* phase for this reduce task can be performed, and then reduce computations can be started. In other words, the computations from three different phases – *Map*, *RSplit*, and *Reduce* – can be performed in parallel, provided the involved data are independent. Therefore, the proposed architectural solution weakens blocking between the stages and, as a result, achieves a higher degree of parallelisation. The overall refinement structure of the partially blocking model is presented on Fig.9.

4.3 Discussion and Future Work

To verify correctness of the presented models, we have discharged around 270 proof obligations for the first formal development, as well as more than 300 for the second one. Approximately 93% of them have been proved automatically by the Rodin platform and the rest have been proved manually in the Rodin interactive proving environment. With help of the ERS approach, we have decomposed the atomicity of the MapReduce framework and hereby achieved

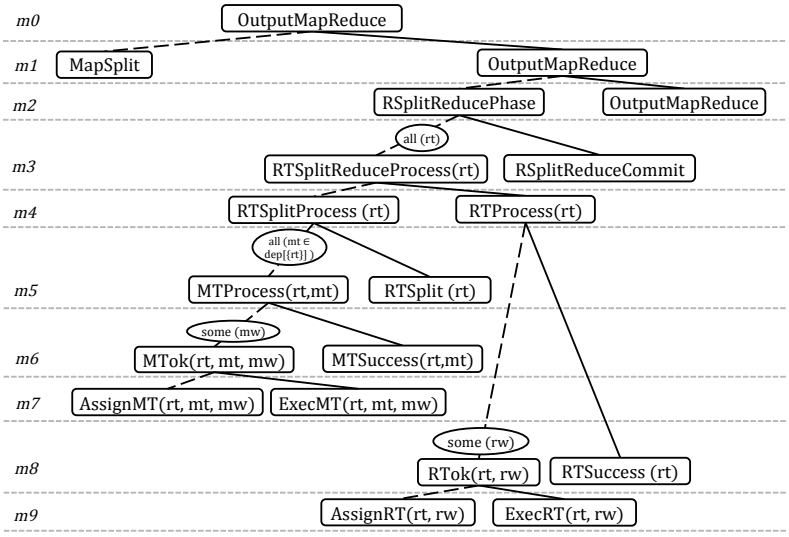


Figure 9: MapReduce Event Refinement Structure: partially blocking model

a higher degree of automation in proving. Moreover, the ERS diagrammatic notation have provided us with additional support to represent the model control flow at different abstraction levels and also simplified reasoning about possible refinement strategies. The whole development and proving effort has taken about one person-month.

As a result of the presented refinement chains, we have arrived at two different centralised Event-B models of the distributed MapReduce framework. As a part of the future work, we are planing to derive distributed models by employing the existing decomposition mechanisms of Event-B. This would result in creating separate formal specifications of the involved software components of the MapReduce framework (such as master, map worker, reduce worker, etc.).

The static part of the modelled system is formally defined in the context component. The definitions of static data structures in the context are mostly very abstract, i.e. they state only essential properties to be satisfied. This makes them generic parameters of the whole formal development. In its turn, such formal development becomes generic, representing a family of the systems that can be described by providing suitable concrete values for the generic parameters. As a continuation of this work, it would be interesting to create formal models for a concrete MapReduce implementation, e.g., the word counting example, by using the Event-B generic instantiation plug-in.

Moreover, to analyse the quantitative characteristics of the proposed models, we are planing to use the Uppaal-SMC model checker. This would allow us to, e.g., assign different data processing rates for the map and reduce tasks and then compare the execution time estimations of two considered architectures.

5 Related Work and Conclusions

The problem of formalisation of the MapReduce framework has been studied in [12]. The authors present a formal model of MapReduce using the CSP method. In their work, they focus on formalising the essential components of the MapReduce framework: the master, mapper, reducer, the underlying file system, and their interactions. In contrast, our focus is on modelling the overall flow of control as well as the data interdependencies between the MapReduce computational phases. Moreover, our approach is based on the stepwise refinement technique that allowed us to gradually unfold the complexity of the MapReduce framework.

Formalisation of MapReduce in Haskel is presented in [9]. Similarly to our approach, it focuses on the program skeleton that underlies MapReduce computations and considers the opportunities for parallelism in executing MapReduce computations. However, in addition to that, we also reason about the involved software components – the master, map and reduce workers – that are associated with the respective map and reduce tasks.

The work [8] presents two approaches based on Coq and JML to formally verify the actual running code of the selected Hadoop MapReduce application. In our work we are more interested in formalisation of MapReduce computations and gradual building of different MapReduce models that are correct-by-construction. The performance issues of MapReduce computations have been studied in the paper [4], focusing on one particular implementation of MapReduce framework – Hadoop. In contrast, we have tried to formally investigate the data interdependencies between the MapReduce phases and their effect on the degree of parallelisation, independently of a concrete MapReduce implementation.

In this paper we have proposed an approach to formalising the MapReduce framework. Our main technical contribution of this paper is two-fold. On the one hand, based on our definition of interdependencies between the processed data as well as the map and reduce stages, we have derived the conditions under which blocking between the stages can be relaxed. Therefore, we have rigorously derived constraints for implementing MapReduce with a higher degree of parallelisation. On the other hand, we have demonstrated how to use the Event Refinement Structure (ERS) technique to formally derive and verify a model of a complex system with a massively parallel architecture and complex dynamic behaviour.

The stepwise refinement approach to deriving a complex system model has demonstrated good scalability and allowed us to express system properties at different levels of abstraction and with a different degree of granularity. Moreover, combining the refinement technique with tool-assisted mathematical proofs have provided us with a scalable approach to verification of a complex system model.

References

[1] J.-R. Abrial. *Modeling in Event-B*. Cambridge University Press, 2010.

[2] Dhruba Borthakur. The Hadoop Distributed File System: Architecture and Design. In *The Apache Software Foundation*, 2007.

[3] Michael Butler. Decomposition Structures for Event-B. In *Integrated Formal Methods, 7th International Conference, IFM 2009*, pages 20–38. Springer Heidelberg, 2009.

[4] Tyson Condie, Neil Conway, Peter Alvaro, and Joseph M. Hellerstein. MapReduce Online. In *NSDI 2010*, pages 20–20. USENIX Association, 2010.

[5] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, pages 137–150. USENIX Association, 2004.

[6] Asieh Salehi Fathabadi, Michael Butler, and Abdolbaghi Rezazadeh. A Systematic Approach to Atomicity Decomposition in Event-B. In *SEFM 2012*, volume 7504 of *LNCS*, pages 78–93. Springer-Verlag Berlin Heidelberg, 2012.

[7] Alan F. Gates, Olga Natkovich, Shubham Chopra, Pradeep Kamath, Shravan M. Narayanamurthy, Christopher Olston, Benjamin Reed, Santhosh Srinivasan, and Utkarsh Srivastava. Building a High-level Dataflow System on Top of Map-Reduce: the Pig Experience. In *Proc. VLDB Endowment*, volume 2, pages 1414–1425, 2009.

[8] Kosuke Ono, Yoichi Hirai, Yoshinori Tanabe, Natsuko Noda, and Masami Hagiya. Using Coq in Specification and Program Extraction of Hadoop MapReduce Applications. In *SEFM 2011*, pages 350–365. Springer Berlin Heidelberg, 2011.

[9] Ralf Lämmel. Google's MapReduce programming model. volume 70, pages 1–30, 2008.

[10] Rodin. Event-B Platform, online at http://www.event-b.org/.

[11] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive - A Warehousing Solution Over a Map-Reduce Framework. In *Proc. VLDB Endowment*, volume 2, pages 1626–1629, 2009.

[12] Fan Yang, Wen Su, Huibiao Zhu, and Qin Li. Formalizing MapReduce with CSP. In *17th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, pages 358–367. IEEE computer society, 2010.

Appendix

Event-B Development

---

1. MapReduce: Blocking Model

CONTEXT MapReduce1_cnt

SETS

- `IData` the input data to be processed
- `OData` the resulting output data
- `MTask` a set of all possible map tasks
- `MData` an abstract type defining the data obtained after map splitting
- `RTask` a set of all possible reduce tasks
- `RData` an abstract type defining the intermediate data obtained during the Map phase
- `MW` a set of map workers
- `RW` a set of reduce workers

CONSTANTS

- `MSplit`
- `Map`
- `RSplit`
- `Reduce`
- `Combine`
- `mnum`
- `rnum`
- `map` the user-supplied function map
- `reduce` the user-supplied function reduce
- `mtasks`
- `rtasks`

AXIOMS

- `axm1` : $finite(IData)$
- `axm2` : $finite(MTask)$
- `axm3` : $finite(MData)$
- `axm4` : $finite(RData)$
- `axm5` : $finite(RTask)$
- `axm6` : $finite(MW)$
- `axm7` : $finite(RW)$

axm8 : $MSplit \in IData \rightarrow (MTask \nrightarrow MData)$

axm9 : $Map \in (MTask \nrightarrow MData) \rightarrow \mathbb{P}_1(MTask \times RData)$

axm10 : $RSplit \in \mathbb{P}_1(MTask \times RData) \rightarrow (RTask \nrightarrow \mathbb{P}_1(RData))$

axm11 : $Reduce \in (RTask \nrightarrow \mathbb{P}_1(RData)) \rightarrow \mathbb{P}_1(OData)$

axm12 : $Combine \in \mathbb{P}_1(OData) \rightarrow OData$

axm13 : $\forall f \cdot f \in MTask \nrightarrow MData \Rightarrow finite(f)$

axm14 : $\forall f \cdot f \in RTask \nrightarrow \mathbb{P}_1(RData) \Rightarrow finite(f)$

axm15 : $\forall f \cdot f \in MTask \nrightarrow MData \Rightarrow dom(f) = dom(Map(f))$

axm16 : $\forall f \cdot f \in \mathbb{P}_1(MTask \times RData) \Rightarrow ran(f) = (\bigcup rt \cdot rt \in dom(RSplit(f)) | RSplit(f)(rt))$

axm17 : $mtasks \in IData \rightarrow \mathbb{P}_1(MTask)$

axm18 : $mnum \in IData \rightarrow \mathbb{N}_1$

axm19 : $\forall idata \cdot idata \in IData \Rightarrow card(MSplit(idata)) = mnum(idata)$

axm20 : $\forall idata \cdot idata \in IData \Rightarrow mtasks(idata) = dom(MSplit(idata))$

axm21 : $\forall idata \cdot idata \in IData \Rightarrow card(mtasks(idata)) = mnum(idata)$

axm22 : $rtasks \in IData \rightarrow \mathbb{P}_1(RTask)$

axm23 : $rnum \in IData \rightarrow \mathbb{N}_1$

axm24 : $\forall idata \cdot idata \in IData \Rightarrow card(RSplit(Map(MSplit(idata)))) = rnum(idata)$

axm25 : $\forall idata \cdot idata \in IData \Rightarrow rtasks(idata) = dom(RSplit(Map(MSplit(idata))))$

axm26 : $\forall idata \cdot idata \in IData \Rightarrow card(rtasks(idata)) = rnum(idata)$

axm27 : $map \in MData \rightarrow \mathbb{P}_1(RData)$

axm28 : $\forall idata, f \cdot idata \in IData \wedge f \subseteq MSplit(idata) \Rightarrow Map(f) = (\bigcup mt \cdot mt \in dom(f) | \{mt\} \times map(f(mt)))$

axm29 : $reduce \in \mathbb{P}_1(RData) \rightarrow \mathbb{P}_1(OData)$

axm30 : $\forall idata, f \cdot idata \in IData \wedge f \subseteq RSplit(Map(MSplit(idata))) \Rightarrow Reduce(f) = (\bigcup rt \cdot rt \in dom(f) | reduce(f(rt)))$

END

MACHINE MapReduce1_m0

SEES MapReduce1_cnt

VARIABLES

output

idata

done

INVARIANTS

inv1 : $output \in OData$

inv2 : $idata \in IData$

inv3 : $done \in BOOL$

inv4 : $done = TRUE \Rightarrow output = Combine(Reduce(RSplit(Map(MSplit(idata)))))$

EVENTS

Initialisation

begin

act1 : $output :\in OData$

act2 : $idata :\in IData$

act3 : $done := FALSE$

end

Event $OutputMapReduce \mathrel{\widehat{=}}$

any

$t1$

$t2$

$t3$

$t4$

$pmtask$

$prtask$

where

grd1 : $pmtask \subseteq MTask$

grd2 : $prtask \subseteq RTask$

grd3 : $t1 \in pmtask \rightarrow MData$

grd4 : $t2 \in \mathbb{P}_1(pmtask \times RData)$

grd5 : $t3 \in prtask \rightarrow \mathbb{P}_1(RData)$

grd6 : $t4 \in \mathbb{P}_1(OData)$

grd7 : $t1 = MSplit(idata)$

grd8 : $t2 = Map(t1)$

grd9 : $t3 = RSplit(t2)$

grd10 : $t4 = Reduce(t3)$

then

act1 : $output := Combine(t4)$

act2 : $done := TRUE$

end

END

MACHINE MapReduce1_m1

REFINES MapReduce1_m0

SEES MapReduce1_cnt

VARIABLES

`output`

`idata`

`MapSplit`

`OutputMapReduce`

`msplit`

INVARIANTS

`inv1` : $MapSplit \in BOOL$

`inv2` : $OutputMapReduce \in BOOL$

`inv3` : $OutputMapReduce = done$

`inv4` : $OutputMapReduce = TRUE \Rightarrow MapSplit = TRUE$

`inv5` : $msplit \in MTask \nrightarrow MData$

`inv6` : $MapSplit = TRUE \Rightarrow msplit = MSplit(idata)$

`inv7` : $MapSplit = TRUE \Rightarrow card(msplit) = mnum(idata)$

`inv8` : $MapSplit = FALSE \Rightarrow msplit = \varnothing$

EVENTS

Initialisation

begin

`act1` : $output :\in OData$

`act2` : $idata :\in IData$

`act3` : $MapSplit := FALSE$

`act4` : $OutputMapReduce := FALSE$

`act5` : $msplit := \varnothing$

end

Event $MapSplit \mathrel{\widehat{=}}$

when

`grd1` : $MapSplit = FALSE$

then

`act1` : $MapSplit := TRUE$

`act2` : $msplit := MSplit(idata)$

end

Event $OutputMapReduce \mathrel{\widehat{=}}$

refines $OutputMapReduce$

any

$t2$

$t3$

$t4$

$prtask$

where

grd1 : $MapSplit = TRUE$

grd2 : $OutputMapReduce = FALSE$

grd3 : $prtask \subseteq RTask$

grd4 : $t2 \in \mathbb{P}_1(dom(msplit) \times RData)$

grd5 : $t3 \in prtask \rightarrow \mathbb{P}_1(RData)$

grd6 : $t4 \in \mathbb{P}_1(OData)$

grd7 : $t2 = Map(msplit)$

grd8 : $t3 = RSplit(t2)$

grd9 : $t4 = Reduce(t3)$

with

pmtask : $\texttt{pmtask} = \texttt{dom}(\texttt{msplit})$

t1 : $\texttt{t1} = \texttt{msplit}$

then

act1 : $output := Combine(t4)$

act2 : $OutputMapReduce := TRUE$

end

END

MACHINE MapReduce1_m2

REFINES MapReduce1_m1

SEES MapReduce1_cnt

VARIABLES

`MapSplit`

`OutputMapReduce`

`idata`

`msplit`

`output`

`MapPhase`

`map_result`

INVARIANTS

inv1 : $MapPhase \in BOOL$

inv2 : $OutputMapReduce = TRUE \Rightarrow MapPhase = TRUE$

inv3 : $MapPhase = TRUE \Rightarrow MapSplit = TRUE$

inv4 : $map_result \in \mathbb{P}(MTask \times RData)$

inv5 : $MapPhase = TRUE \Rightarrow map_result \in \mathbb{P}_1(dom(msplit) \times RData)$

inv6 : $MapPhase = TRUE \Rightarrow map_result = Map(msplit)$

inv7 : $MapSplit = FALSE \Rightarrow map_result = \varnothing$

EVENTS

Initialisation

extended

begin

act1 : `output` $:\in$ `OData`

act2 : `idata` $:\in$ `IData`

act3 : `MapSplit` $:=$ `FALSE`

act4 : `OutputMapReduce` $:=$ `FALSE`

act5 : `msplit` $:= \varnothing$

act6 : $map_result := \varnothing$

act7 : $MapPhase := FALSE$

end

Event $MapSplit \mathrel{\widehat{=}}$

extends *MapSplit*

when

grd1 : `MapSplit` $=$ `FALSE`

then

act1 : `MapSplit` $:=$ `TRUE`

act2 : `msplit` $:=$ `MSplit(idata)`

end

Event $MapPhase \mathrel{\widehat{=}}$

when

grd1 : $MapSplit = TRUE$
grd2 : $MapPhase = FALSE$

then

act1 : $MapPhase := TRUE$
act2 : $map_result := Map(msplit)$

end

Event $OutputMapReduce \widehat{=}$

refines $OutputMapReduce$

any

$t3$
$t4$
$prtask$

where

grd1 : $MapPhase = TRUE$
grd2 : $OutputMapReduce = FALSE$
grd3 : $prtask \subseteq RTask$
grd4 : $t3 \in prtask \rightarrow \mathbb{P}_1(RData)$
grd5 : $t4 \in \mathbb{P}_1(OData)$
grd6 : $t3 = RSplit(map_result)$
grd7 : $t4 = Reduce(t3)$

with

t2 : `t2 = map_result`

then

act1 : $output := Combine(t4)$
act2 : $OutputMapReduce := TRUE$

end

END

MACHINE MapReduce1_m3

REFINES MapReduce1_m2

SEES MapReduce1_cnt

VARIABLES

`MapSplit`

`OutputMapReduce`

`idata`

`msplit`

`output`

`MapPhase`

`map_result`

`ReduceSplit`

`rsplit`

INVARIANTS

`inv1` : $ReduceSplit \in BOOL$

`inv2` : $ReduceSplit = TRUE \Rightarrow MapPhase = TRUE$

`inv3` : $OutputMapReduce = TRUE \Rightarrow ReduceSplit = TRUE$

`inv4` : $rsplit \in RTask \pfun \mathbb{P}_1(RData)$

`inv5` : $ReduceSplit = TRUE \Rightarrow rsplit = RSplit(map_result)$

`inv6` : $ReduceSplit = TRUE \Rightarrow card(rsplit) = rnum(idata)$

`inv7` : $ReduceSplit = FALSE \Rightarrow rsplit = \varnothing$

EVENTS

Initialisation

extended

begin

`act1` : `output` $:\in$ `OData`

`act2` : `idata` $:\in$ `IData`

`act3` : `MapSplit` $:=$ `FALSE`

`act4` : `OutputMapReduce` $:=$ `FALSE`

`act5` : `msplit` $:= \varnothing$

`act6` : `map_result` $:= \varnothing$

`act7` : `MapPhase` $:=$ `FALSE`

`act8` : $ReduceSplit := FALSE$

`act9` : $rsplit := \varnothing$

end

Event $MapSplit \mathrel{\widehat{=}}$

extends *MapSplit*

when

`grd1` : `MapSplit` $=$ `FALSE`

then

`act1` : `MapSplit` $:=$ `TRUE`

act2 : msplit := MSplit(idata)

end

Event *MapPhase* $\widehat{=}$

extends *MapPhase*

when

grd1 : MapSplit = TRUE

grd2 : MapPhase = FALSE

then

act1 : MapPhase := TRUE

act2 : map_result := Map(msplit)

end

Event *ReduceSplit* $\widehat{=}$

when

grd1 : $MapPhase = TRUE$

grd2 : $ReduceSplit = FALSE$

then

act1 : $ReduceSplit := TRUE$

act2 : $rsplit := RSplit(map_result)$

end

Event *OutputMapReduce* $\widehat{=}$

refines *OutputMapReduce*

any

$t4$

where

grd1 : $ReduceSplit = TRUE$

grd2 : $OutputMapReduce = FALSE$

grd3 : $t4 \in \mathbb{P}_1(OData)$

grd4 : $t4 = Reduce(rsplit)$

with

t3 : t3 = rsplit

prtask : prtask = dom(rsplit)

then

act1 : $output := Combine(t4)$

act2 : $OutputMapReduce := TRUE$

end

END

MACHINE MapReduce1_m4

REFINES MapReduce1_m3

SEES MapReduce1_cnt

VARIABLES

MapPhase

MapSplit

OutputMapReduce

ReduceSplit

idata

map_result

msplit

output

rsplit

ReducePhase

reduce_result

INVARIANTS

inv1 : $ReducePhase \in BOOL$

inv2 : $OutputMapReduce = TRUE \Rightarrow ReducePhase = TRUE$

inv3 : $ReducePhase = TRUE \Rightarrow ReduceSplit = TRUE$

inv4 : $reduce_result \in \mathbb{P}(OData)$

inv5 : $ReducePhase = TRUE \Rightarrow reduce_result \in \mathbb{P}_1(OData)$

inv6 : $ReducePhase = TRUE \Rightarrow reduce_result = Reduce(rsplit)$

inv7 : $ReducePhase = FALSE \Rightarrow reduce_result = \varnothing$

EVENTS

Initialisation

extended

begin

act1 : output $:\in$ OData

act2 : idata $:\in$ IData

act3 : MapSplit $:=$ FALSE

act4 : OutputMapReduce $:=$ FALSE

act5 : msplit $:= \varnothing$

act6 : map_result $:= \varnothing$

act7 : MapPhase $:=$ FALSE

act8 : ReduceSplit $:=$ FALSE

act9 : rsplit $:= \varnothing$

act10 : $ReducePhase := FALSE$

act11 : $reduce_result := \varnothing$

end

Event *MapSplit* $\widehat{=}$

extends *MapSplit*

when
grd1 : MapSplit = FALSE
then
act1 : MapSplit := TRUE
act2 : msplit := MSplit(idata)
end
Event *MapPhase* $\widehat{=}$
extends *MapPhase*
when
grd1 : MapSplit = TRUE
grd2 : MapPhase = FALSE
then
act1 : MapPhase := TRUE
act2 : map_result := Map(msplit)
end
Event *ReduceSplit* $\widehat{=}$
extends *ReduceSplit*
when
grd1 : MapPhase = TRUE
grd2 : ReduceSplit = FALSE
then
act1 : ReduceSplit := TRUE
act2 : rsplit := RSplit(map_result)
end
Event *ReducePhase* $\widehat{=}$
when
grd1 : $ReduceSplit = TRUE$
grd2 : $ReducePhase = FALSE$
then
act1 : $ReducePhase := TRUE$
act2 : $reduce_result := Reduce(rsplit)$
end
Event *OutputMapReduce* $\widehat{=}$
refines *OutputMapReduce*
when
grd1 : $ReducePhase = TRUE$
grd2 : $OutputMapReduce = FALSE$
with
t4 : t4 = reduce_result
then
act1 : $output := Combine(reduce_result)$
act2 : $OutputMapReduce := TRUE$
end
END

MACHINE MapReduce1_m5

REFINES MapReduce1_m4

SEES MapReduce1_cnt

VARIABLES

`MapCommit`

`MapSplit`

`OutputMapReduce`

`ReducePhase`

`ReduceSplit`

`idata`

`map_result`

`msplit`

`output`

`reduce_result`

`rsplit`

`MTProcess`

`MTProcess_result`

INVARIANTS

`inv1` : $MapCommit \in BOOL$

`inv2` : $MTProcess \subseteq dom(msplit)$

`inv3` : $MTProcess \neq \varnothing \Rightarrow MapSplit = TRUE$

`inv4` : $MapCommit = TRUE \Rightarrow MTProcess = dom(msplit)$

`inv5` : $MapCommit = MapPhase$

`inv6` : $MapCommit = TRUE \Rightarrow MapSplit = TRUE$

`inv7` : $ReduceSplit = TRUE \Rightarrow MapCommit = TRUE$

`inv8` : $MTProcess_result \in MTask \pfun \mathbb{P}_1(RData)$

`inv9` : $MTProcess = dom(MTProcess_result)$

`inv10` : $\forall mt \cdot mt \in dom(MTProcess_result) \Rightarrow MTProcess_result(mt) = map(msplit(mt))$

EVENTS

Initialisation

begin

`act1` : $output :\in OData$

`act2` : $idata :\in IData$

`act3` : $MapSplit := FALSE$

`act4` : $OutputMapReduce := FALSE$

`act5` : $msplit := \varnothing$

`act6` : $map_result := \varnothing$

`act7` : $MapCommit := FALSE$

`act8` : $ReduceSplit := FALSE$

`act9` : $rsplit := \varnothing$

act10 : $ReducePhase := FALSE$
act11 : $reduce_result := \varnothing$
act12 : $MTProcess := \varnothing$
act13 : $MTProcess_result := \varnothing$

end

Event *MapSplit* $\widehat{=}$

extends *MapSplit*

when

grd1 : `MapSplit` = `FALSE`

then

act1 : `MapSplit` := `TRUE`
act2 : `msplit` := `MSplit(idata)`

end

Event *MTProcess* $\widehat{=}$

any

mt

where

grd1 : $MapSplit = TRUE$
grd2 : $mt \in dom(msplit)$
grd3 : $mt \notin MTProcess$

then

act1 : $MTProcess := MTProcess \cup \{mt\}$
act2 : $MTProcess_result(mt) := map(msplit(mt))$

end

Event *MapCommit* $\widehat{=}$

refines *MapPhase*

when

grd1 : $MapSplit = TRUE$
grd2 : $MapCommit = FALSE$
grd3 : $MTProcess = dom(msplit)$

then

act1 : $MapCommit := TRUE$
act2 : $map_result := (\bigcup mt \cdot mt \in dom(msplit) | \{mt\} \times MTProcess_result(mt))$

end

Event *ReduceSplit* $\widehat{=}$

refines *ReduceSplit*

when

grd1 : $MapCommit = TRUE$
grd2 : $ReduceSplit = FALSE$

then

act1 : $ReduceSplit := TRUE$
act2 : $rsplit := RSplit(map_result)$

end

Event *ReducePhase* $\widehat{=}$
extends *ReducePhase*
when
grd1 : ReduceSplit = TRUE
grd2 : ReducePhase = FALSE
then
act1 : ReducePhase := TRUE
act2 : reduce_result := Reduce(rsplit)
end
Event *OutputMapReduce* $\widehat{=}$
extends *OutputMapReduce*
when
grd1 : ReducePhase = TRUE
grd2 : OutputMapReduce = FALSE
then
act1 : output := Combine(reduce_result)
act2 : OutputMapReduce := TRUE
end
END

MACHINE MapReduce1_m6

REFINES MapReduce1_m5

SEES MapReduce1_cnt

VARIABLES

`MapCommit`

`MTProcess_result`

`MapSplit`

`OutputMapReduce`

`ReducePhase`

`ReduceSplit`

`idata`

`map_result`

`msplit`

`output`

`reduce_result`

`rsplit`

`MTok`

`MTSuccess`

`mworker_result`

INVARIANTS

`inv1` : $MTok \subseteq dom(msplit) \times MW$

`inv2` : $MTSuccess \subseteq dom(MTok)$

`inv3` : $MTSuccess = MTProcess$

`inv4` : $MTok \neq \varnothing \Rightarrow MapSplit = TRUE$

`inv5` : $mworker_result \in MW \pfun (MTask \pfun \mathbb{P}_1(RData))$

`inv6` : $MapSplit = FALSE \Rightarrow dom(mworker_result) = \varnothing$

`inv7` : $dom(mworker_result) = ran(MTok)$

`inv8` : $\forall mt, mw \cdot mt \mapsto mw \in MTok \Rightarrow mw \in dom(mworker_result) \land mt \in dom(mworker_result(mw))$

`inv9` : $\forall mw, mt \cdot mt \mapsto mw \in MTok \Rightarrow mworker_result(mw)(mt) = map(msplit(mt))$

EVENTS

Initialisation

begin

`act1` : $output :\in OData$

`act2` : $idata :\in IData$

`act3` : $MapSplit := FALSE$

`act4` : $OutputMapReduce := FALSE$

`act5` : $msplit := \varnothing$

`act6` : $map_result := \varnothing$

`act7` : $MapCommit := FALSE$

act8 : $ReduceSplit := FALSE$
act9 : $rsplit := \varnothing$
act10 : $ReducePhase := FALSE$
act11 : $reduce_result := \varnothing$
act12 : $MTSuccess := \varnothing$
act13 : $MTProcess_result := \varnothing$
act14 : $MTok := \varnothing$
act15 : $mworker_result := \varnothing$

end

Event $MapSplit \widehat{=}$

extends $MapSplit$

when

grd1 : $\texttt{MapSplit} = \texttt{FALSE}$

then

act1 : $\texttt{MapSplit} := \texttt{TRUE}$
act2 : $\texttt{msplit} := \texttt{MSplit}(\texttt{idata})$

end

Event $MTok \widehat{=}$

any

mt
mw

where

grd1 : $MapSplit = TRUE$
grd2 : $mt \in dom(msplit)$
grd3 : $mw \in MW$
grd4 : $mt \mapsto mw \notin MTok$
grd6 : $mw \notin dom(mworker_result)$

then

act1 : $MTok := MTok \cup \{mt \mapsto mw\}$
act2 : $mworker_result := mworker_result \cup \{mw \mapsto \{mt \mapsto map(msplit(mt))\}\}$

end

Event $MTSuccess \widehat{=}$

refines $MTProcess$

any

mt
mw

where

grd1 : $mt \in dom(msplit)$
grd2 : $mt \notin MTSuccess$
grd3 : $mt \mapsto mw \in MTok$

then

act1 : $MTSuccess := MTSuccess \cup \{mt\}$
act2 : $MTProcess_result(mt) := mworker_result(mw)(mt)$

end

Event *MapCommit* $\widehat{=}$

refines *MapCommit*

when

grd1 : $MapSplit = TRUE$

grd2 : $MapCommit = FALSE$

grd3 : $MTSuccess = dom(msplit)$

then

act1 : $MapCommit := TRUE$

act2 : $map_result := (\bigcup mt \cdot mt \in dom(msplit) | \{mt\} \times MTProcess_result(mt))$

end

Event *ReduceSplit* $\widehat{=}$

extends *ReduceSplit*

when

grd1 : $\texttt{MapCommit} = \texttt{TRUE}$

grd2 : $\texttt{ReduceSplit} = \texttt{FALSE}$

then

act1 : $\texttt{ReduceSplit} := \texttt{TRUE}$

act2 : $\texttt{rsplit} := \texttt{RSplit}(\texttt{map_result})$

end

Event *ReducePhase* $\widehat{=}$

extends *ReducePhase*

when

grd1 : $\texttt{ReduceSplit} = \texttt{TRUE}$

grd2 : $\texttt{ReducePhase} = \texttt{FALSE}$

then

act1 : $\texttt{ReducePhase} := \texttt{TRUE}$

act2 : $\texttt{reduce_result} := \texttt{Reduce}(\texttt{rsplit})$

end

Event *OutputMapReduce* $\widehat{=}$

extends *OutputMapReduce*

when

grd1 : $\texttt{ReducePhase} = \texttt{TRUE}$

grd2 : $\texttt{OutputMapReduce} = \texttt{FALSE}$

then

act1 : $\texttt{output} := \texttt{Combine}(\texttt{reduce_result})$

act2 : $\texttt{OutputMapReduce} := \texttt{TRUE}$

end

END

MACHINE MapReduce1_m7

REFINES MapReduce1_m6

SEES MapReduce1_cnt

VARIABLES

`MapCommit`
`MTProcess_result`
`MapSplit`
`OutputMapReduce`
`ReducePhase`
`ReduceSplit`
`idata`
`map_result`
`msplit`
`output`
`reduce_result`
`rsplit`
`ExecMT`
`MTSuccess`
`mworker_result`
`AssignMT`

INVARIANTS

`inv1` : $AssignMT \subseteq dom(msplit) \times MW$
`inv2` : $ExecMT = MTok$
`inv3` : $ExecMT \subseteq AssignMT$
`inv4` : $AssignMT \neq \varnothing \Rightarrow MapSplit = TRUE$
`inv5` : $MTSuccess \subseteq dom(ExecMT)$

EVENTS

Initialisation

begin

`act1` : $output :\in OData$
`act2` : $idata :\in IData$
`act3` : $MapSplit := FALSE$
`act4` : $OutputMapReduce := FALSE$
`act5` : $msplit := \varnothing$
`act6` : $map_result := \varnothing$
`act7` : $MapCommit := FALSE$
`act8` : $ReduceSplit := FALSE$
`act9` : $rsplit := \varnothing$
`act10` : $ReducePhase := FALSE$
`act11` : $reduce_result := \varnothing$
`act12` : $MTSuccess := \varnothing$

act13 : $MTProcess_result := \varnothing$
act14 : $ExecMT := \varnothing$
act15 : $mworker_result := \varnothing$
act16 : $AssignMT := \varnothing$

end

Event $MapSplit \widehat{=}$

extends $MapSplit$

when

grd1 : $\texttt{MapSplit} = \texttt{FALSE}$

then

act1 : $\texttt{MapSplit} := \texttt{TRUE}$
act2 : $\texttt{msplit} := \texttt{MSplit(idata)}$

end

Event $AssignMT \widehat{=}$

any

mw
mt

where

grd1 : $MapSplit = TRUE$
grd2 : $mt \in dom(msplit)$
grd3 : $mw \in MW$
grd4 : $mt \mapsto mw \notin AssignMT$
grd5 : $mw \notin ran(AssignMT)$

then

act1 : $AssignMT := AssignMT \cup \{mt \mapsto mw\}$

end

Event $ExecMT \widehat{=}$

refines $MTok$

any

mt
mw

where

grd1 : $MapSplit = TRUE$
grd2 : $mt \in dom(msplit)$
grd3 : $mw \in MW$
grd4 : $mt \mapsto mw \notin ExecMT$
grd6 : $mw \notin dom(mworker_result)$
grd7 : $mt \mapsto mw \in AssignMT$

then

act1 : $ExecMT := ExecMT \cup \{mt \mapsto mw\}$
act2 : $mworker_result := mworker_result \cup \{mw \mapsto \{mt \mapsto map(msplit(mt))\}\}$

end

Event $MTSuccess \widehat{=}$

extends *MTSuccess*

any

mt

mw

where

grd1 : $mt \in dom(msplit)$

grd2 : $mt \notin MTSuccess$

grd3 : $mt \mapsto mw \in ExecMT$

then

act1 : $MTSuccess := MTSuccess \cup \{mt\}$

act2 : $MTProcess_result(mt) := mworker_result(mw)(mt)$

end

Event *MapCommit* $\widehat{=}$

extends *MapCommit*

when

grd1 : $MapSplit = TRUE$

grd2 : $MapCommit = FALSE$

grd3 : $MTSuccess = dom(msplit)$

then

act1 : $MapCommit := TRUE$

act2 : $map_result := (\bigcup mt \cdot mt \in dom(msplit) | \{mt\} \times MTProcess_result(mt))$

end

Event *ReduceSplit* $\widehat{=}$

extends *ReduceSplit*

when

grd1 : $MapCommit = TRUE$

grd2 : $ReduceSplit = FALSE$

then

act1 : $ReduceSplit := TRUE$

act2 : $rsplit := RSplit(map_result)$

end

Event *ReducePhase* $\widehat{=}$

extends *ReducePhase*

when

grd1 : $ReduceSplit = TRUE$

grd2 : $ReducePhase = FALSE$

then

act1 : $ReducePhase := TRUE$

act2 : $reduce_result := Reduce(rsplit)$

end

Event *OutputMapReduce* $\widehat{=}$

extends *OutputMapReduce*

when

```
        grd1 : ReducePhase = TRUE
        grd2 : OutputMapReduce = FALSE
    then
        act1 : output := Combine(reduce_result)
        act2 : OutputMapReduce := TRUE
    end
END
```

MACHINE MapReduce1_m8

REFINES MapReduce1_m7

SEES MapReduce1_cnt

VARIABLES

`MapCommit`
`MTProcess_result`
`MapSplit`
`OutputMapReduce`
`ReduceCommit`
`ReduceSplit`
`idata`
`map_result`
`msplit`
`output`
`reduce_result`
`rsplit`
`ExecMT`
`MTSuccess`
`mworker_result`
`AssignMT`
`RTProcess`
`RTProcess_result`

INVARIANTS

inv1 : $ReduceCommit \in BOOL$
inv2 : $ReduceCommit = ReducePhase$
inv3 : $RTProcess \subseteq dom(rsplit)$
inv4 : $ReduceCommit = TRUE \Rightarrow ReduceSplit = TRUE$
inv5 : $OutputMapReduce = TRUE \Rightarrow ReduceCommit = TRUE$
inv6 : $RTProcess \neq \varnothing \Rightarrow ReduceSplit = TRUE$
inv7 : $ReduceCommit = TRUE \Rightarrow RTProcess = dom(rsplit)$
inv8 : $RTProcess_result \in RTask \pfun \mathbb{P}_1(OData)$
inv9 : $RTProcess = dom(RTProcess_result)$
inv10 : $\forall rt \cdot rt \in dom(RTProcess_result) \Rightarrow RTProcess_result(rt) = reduce(rsplit(rt))$
inv11 : $ReduceSplit = FALSE \Rightarrow dom(RTProcess_result) = \varnothing$

EVENTS

Initialisation

begin

act1 : $output :\in OData$
act2 : $idata :\in IData$

act3 : $MapSplit := FALSE$
act4 : $OutputMapReduce := FALSE$
act5 : $msplit := \varnothing$
act6 : $map_result := \varnothing$
act7 : $MapCommit := FALSE$
act8 : $ReduceSplit := FALSE$
act9 : $rsplit := \varnothing$
act10 : $ReduceCommit := FALSE$
act11 : $reduce_result := \varnothing$
act12 : $MTSuccess := \varnothing$
act13 : $MTProcess_result := \varnothing$
act14 : $ExecMT := \varnothing$
act15 : $mworker_result := \varnothing$
act16 : $AssignMT := \varnothing$
act17 : $RTProcess := \varnothing$
act18 : $RTProcess_result := \varnothing$

end

Event $MapSplit \widehat{=}$
extends $MapSplit$

when

grd1 : $\texttt{MapSplit} = \texttt{FALSE}$

then

act1 : $\texttt{MapSplit} := \texttt{TRUE}$
act2 : $\texttt{msplit} := \texttt{MSplit}(\texttt{idata})$

end

Event $AssignMT \widehat{=}$
extends $AssignMT$

any

mw
mt

where

grd1 : $\texttt{MapSplit} = \texttt{TRUE}$
grd2 : $\texttt{mt} \in \texttt{dom}(\texttt{msplit})$
grd3 : $\texttt{mw} \in \texttt{MW}$
grd4 : $\texttt{mt} \mapsto \texttt{mw} \notin \texttt{AssignMT}$
grd5 : $\texttt{mw} \notin \texttt{ran}(\texttt{AssignMT})$

then

act1 : $\texttt{AssignMT} := \texttt{AssignMT} \cup \{\texttt{mt} \mapsto \texttt{mw}\}$

end

Event $ExecMT \widehat{=}$
extends $ExecMT$

any

mt
mw

where

grd1 : $\texttt{MapSplit} = \texttt{TRUE}$

grd2 : $\texttt{mt} \in \texttt{dom}(\texttt{msplit})$

grd3 : $\texttt{mw} \in \texttt{MW}$

grd4 : $\texttt{mt} \mapsto \texttt{mw} \notin \texttt{ExecMT}$

grd6 : $\texttt{mw} \notin \texttt{dom}(\texttt{mworker_result})$

grd7 : $\texttt{mt} \mapsto \texttt{mw} \in \texttt{AssignMT}$

then

act1 : $\texttt{ExecMT} := \texttt{ExecMT} \cup \{\texttt{mt} \mapsto \texttt{mw}\}$

act2 : $\texttt{mworker_result} := \texttt{mworker_result} \cup \{\texttt{mw} \mapsto \{\texttt{mt} \mapsto \texttt{map}(\texttt{msplit}(\texttt{mt}))\}\}$

end

Event *MapSuccess* $\widehat{=}$

extends *MapSuccess*

any

mt

mw

where

grd1 : $\texttt{mt} \in \texttt{dom}(\texttt{msplit})$

grd2 : $\texttt{mt} \notin \texttt{MTSuccess}$

grd3 : $\texttt{mt} \mapsto \texttt{mw} \in \texttt{ExecMT}$

then

act1 : $\texttt{MTSuccess} := \texttt{MTSuccess} \cup \{\texttt{mt}\}$

act2 : $\texttt{MTProcess_result}(\texttt{mt}) := \texttt{mworker_result}(\texttt{mw})(\texttt{mt})$

end

Event *MTCommit* $\widehat{=}$

extends *MTCommit*

when

grd1 : $\texttt{MapSplit} = \texttt{TRUE}$

grd2 : $\texttt{MapCommit} = \texttt{FALSE}$

grd3 : $\texttt{MTSuccess} = \texttt{dom}(\texttt{msplit})$

then

act1 : $\texttt{MapCommit} := \texttt{TRUE}$

act2 : $\texttt{map_result} := (\bigcup \texttt{mt} \cdot \texttt{mt} \in \texttt{dom}(\texttt{msplit}) | \{\texttt{mt}\} \times \texttt{MTProcess_result}(\texttt{mt}))$

end

Event *ReduceSplit* $\widehat{=}$

extends *ReduceSplit*

when

grd1 : $\texttt{MapCommit} = \texttt{TRUE}$

grd2 : $\texttt{ReduceSplit} = \texttt{FALSE}$

then

act1 : $\texttt{ReduceSplit} := \texttt{TRUE}$

act2 : $\texttt{rsplit} := \texttt{RSplit}(\texttt{map_result})$

end

Event $RTProcess \mathrel{\widehat{=}}$

any

rt

where

grd1 : $ReduceSplit = TRUE$

grd2 : $rt \in dom(rsplit)$

grd3 : $rt \notin RTProcess$

then

act1 : $RTProcess := RTProcess \cup \{rt\}$

act2 : $RTProcess_result(rt) := reduce(rsplit(rt))$

end

Event $ReduceCommit \mathrel{\widehat{=}}$

refines $ReducePhase$

when

grd1 : $ReduceSplit = TRUE$

grd2 : $ReduceCommit = FALSE$

grd3 : $RTProcess = dom(rsplit)$

then

act1 : $ReduceCommit := TRUE$

act2 : $reduce_result := (\bigcup rt \cdot rt \in dom(rsplit) | RTProcess_result(rt))$

end

Event $OutputMapReduce \mathrel{\widehat{=}}$

refines $OutputMapReduce$

when

grd1 : $ReduceCommit = TRUE$

grd2 : $OutputMapReduce = FALSE$

then

act1 : $output := Combine(reduce_result)$

act2 : $OutputMapReduce := TRUE$

end

END

MACHINE MapReduce1_m9

REFINES MapReduce1_m8

SEES MapReduce1_cnt

VARIABLES

MapCommit
MTProcess_result
MapSplit
OutputMapReduce
ReduceCommit
ReduceSplit
idata
map_result
msplit
output
reduce_result
rsplit
ExecMT
MTSuccess
mworker_result
AssignMT
RTSuccess
RTProcess_result
RTok
rworker_result

INVARIANTS

inv1 : $RTok \subseteq dom(rsplit) \times RW$

inv2 : $RTSuccess \subseteq dom(rsplit)$

inv3 : $RTSuccess \subseteq dom(RTok)$

inv4 : $RTok \neq \varnothing \Rightarrow ReduceSplit = TRUE$

inv5 : $RTSuccess = RTProcess$

inv6 : $rworker_result \in RW \pfun (RTask \pfun \mathbb{P}_1(OData))$

inv7 : $ReduceSplit = FALSE \Rightarrow dom(rworker_result) = \varnothing$

inv8 : $dom(rworker_result) = ran(RTok)$

inv9 : $\forall rt, rw \cdot rt \mapsto rw \in RTok \Rightarrow rw \in dom(rworker_result) \land rt \in dom(rworker_result(rw))$

inv10 : $\forall rw, rt \cdot rt \mapsto rw \in RTok \Rightarrow rworker_result(rw)(rt) = reduce(rsplit(rt))$

inv11 : $dom(RTok) \subseteq dom(rsplit)$

EVENTS

Initialisation

begin

act1 : $output :\in OData$
act2 : $idata :\in IData$
act3 : $MapSplit := FALSE$
act4 : $OutputMapReduce := FALSE$
act5 : $msplit := \varnothing$
act6 : $map_result := \varnothing$
act7 : $MapCommit := FALSE$
act8 : $ReduceSplit := FALSE$
act9 : $rsplit := \varnothing$
act10 : $ReduceCommit := FALSE$
act11 : $reduce_result := \varnothing$
act12 : $MTSuccess := \varnothing$
act13 : $MTProcess_result := \varnothing$
act14 : $ExecMT := \varnothing$
act15 : $mworker_result := \varnothing$
act16 : $AssignMT := \varnothing$
act17 : $RTSuccess := \varnothing$
act18 : $RTProcess_result := \varnothing$
act19 : $rworker_result := \varnothing$
act20 : $RTok := \varnothing$

end

Event $MapSplit \mathrel{\widehat{=}}$

extends $MapSplit$

when

grd1 : $\texttt{MapSplit} = \texttt{FALSE}$

then

act1 : $\texttt{MapSplit} := \texttt{TRUE}$
act2 : $\texttt{msplit} := \texttt{MSplit}(\texttt{idata})$

end

Event $AssignMT \mathrel{\widehat{=}}$

extends $AssignMT$

any

mw
mt

where

grd1 : $\texttt{MapSplit} = \texttt{TRUE}$
grd2 : $\texttt{mt} \in \texttt{dom}(\texttt{msplit})$
grd3 : $\texttt{mw} \in \texttt{MW}$
grd4 : $\texttt{mt} \mapsto \texttt{mw} \notin \texttt{AssignMT}$
grd5 : $\texttt{mw} \notin \texttt{ran}(\texttt{AssignMT})$

then

act1 : $\texttt{AssignMT} := \texttt{AssignMT} \cup \{\texttt{mt} \mapsto \texttt{mw}\}$

end

Event $ExecMT \mathrel{\widehat{=}}$

extends *ExecMT*

any

mt

mw

where

grd1 : MapSplit = TRUE

grd2 : mt ∈ dom(msplit)

grd3 : mw ∈ MW

grd4 : mt ↦ mw ∉ ExecMT

grd6 : mw ∉ dom(mworker_result)

grd7 : mt ↦ mw ∈ AssignMT

then

act1 : ExecMT := ExecMT ∪ {mt ↦ mw}

act2 : mworker_result := mworker_result∪{mw ↦ {mt ↦ map(msplit(mt))}}

end

Event *MTSuccess* ≙

extends *MTSuccess*

any

mt

mw

where

grd1 : mt ∈ dom(msplit)

grd2 : mt ∉ MTSuccess

grd3 : mt ↦ mw ∈ ExecMT

then

act1 : MTSuccess := MTSuccess ∪ {mt}

act2 : MTProcess_result(mt) := mworker_result(mw)(mt)

end

Event *MapCommit* ≙

extends *MapCommit*

when

grd1 : MapSplit = TRUE

grd2 : MapCommit = FALSE

grd3 : MTSuccess = dom(msplit)

then

act1 : MapCommit := TRUE

act2 : map_result := (⋃ mt·mt ∈ dom(msplit)|{mt}×MTProcess_result(mt))

end

Event *ReduceSplit* ≙

extends *ReduceSplit*

when

grd1 : MapCommit = TRUE

grd2 : ReduceSplit = FALSE

then

act1 : `ReduceSplit` := `TRUE`

act2 : `rsplit` := `RSplit(map_result)`

end

Event *RTok* $\widehat{=}$

any

rt

rw

where

grd1 : $ReduceSplit = TRUE$

grd2 : $rt \in dom(rsplit)$

grd3 : $rw \in RW$

grd4 : $rt \mapsto rw \notin RTok$

grd5 : $rw \notin dom(rworker_result)$

then

act1 : $RTok := RTok \cup \{rt \mapsto rw\}$

act2 : $rworker_result := rworker_result \cup \{rw \mapsto \{rt \mapsto reduce(rsplit(rt))\}\}$

end

Event *RTSuccess* $\widehat{=}$

refines *RTProcess*

any

rt

rw

where

grd1 : $ReduceSplit = TRUE$

grd2 : $rt \in dom(rsplit)$

grd3 : $rt \notin RTSuccess$

grd4 : $rt \mapsto rw \in RTok$

then

act1 : $RTSuccess := RTSuccess \cup \{rt\}$

act2 : $RTProcess_result(rt) := rworker_result(rw)(rt)$

end

Event *ReduceCommit* $\widehat{=}$

refines *ReduceCommit*

when

grd1 : $ReduceSplit = TRUE$

grd2 : $ReduceCommit = FALSE$

grd3 : $RTSuccess = dom(rsplit)$

then

act1 : $ReduceCommit := TRUE$

act2 : $reduce_result := (\bigcup rt \cdot rt \in dom(rsplit) | RTProcess_result(rt))$

end

Event *OutputMapReduce* $\widehat{=}$

extends *OutputMapReduce*

when

grd1 : ReduceCommit = TRUE

grd2 : OutputMapReduce = FALSE

then

act1 : output := Combine(reduce_result)

act2 : OutputMapReduce := TRUE

end

END

MACHINE MapReduce1_m_10

REFINES MapReduce1_m9

SEES MapReduce1_cnt

VARIABLES

`MapCommit`
`MTProcess_result`
`MapSplit`
`OutputMapReduce`
`ReduceCommit`
`ReduceSplit`
`idata`
`map_result`
`msplit`
`output`
`reduce_result`
`rsplit`
`ExecMT`
`MTSuccess`
`mworker_result`
`AssignMT`
`RTSuccess`
`RTProcess_result`
`ExecRT`
`rworker_result`
`AssignRT`

INVARIANTS

`inv1` : $AssignRT \subseteq dom(rsplit) \times RW$
`inv2` : $ExecRT = RTok$
`inv3` : $ExecRT \subseteq AssignRT$
`inv4` : $AssignRT \neq \varnothing \Rightarrow ReduceSplit = TRUE$
`inv5` : $RTSuccess \subseteq dom(ExecRT)$

EVENTS

Initialisation

begin

`act1` : $output :\in OData$
`act2` : $idata :\in IData$
`act3` : $MapSplit := FALSE$
`act4` : $OutputMapReduce := FALSE$
`act5` : $msplit := \varnothing$
`act6` : $map_result := \varnothing$

act7 : $MapCommit := FALSE$
act8 : $ReduceSplit := FALSE$
act9 : $rsplit := \varnothing$
act10 : $ReduceCommit := FALSE$
act11 : $reduce_result := \varnothing$
act12 : $MTSuccess := \varnothing$
act13 : $MTProcess_result := \varnothing$
act14 : $ExecMT := \varnothing$
act15 : $mworker_result := \varnothing$
act16 : $AssignMT := \varnothing$
act17 : $RTSuccess := \varnothing$
act18 : $RTProcess_result := \varnothing$
act19 : $rworker_result := \varnothing$
act20 : $ExecRT := \varnothing$
act21 : $AssignRT := \varnothing$

end

Event $MapSplit \widehat{=}$
extends $MapSplit$

when

grd1 : $\texttt{MapSplit} = \texttt{FALSE}$

then

act1 : $\texttt{MapSplit} := \texttt{TRUE}$
act2 : $\texttt{msplit} := \texttt{MSplit}(\texttt{idata})$

end

Event $AssignMT \widehat{=}$
extends $AssignMT$

any

mw
mt

where

grd1 : $\texttt{MapSplit} = \texttt{TRUE}$
grd2 : $\texttt{mt} \in \texttt{dom}(\texttt{msplit})$
grd3 : $\texttt{mw} \in \texttt{MW}$
grd4 : $\texttt{mt} \mapsto \texttt{mw} \notin \texttt{AssignMT}$
grd5 : $\texttt{mw} \notin \texttt{ran}(\texttt{AssignMT})$

then

act1 : $\texttt{AssignMT} := \texttt{AssignMT} \cup \{\texttt{mt} \mapsto \texttt{mw}\}$

end

Event $ExecMT \widehat{=}$
extends $ExecMT$

any

mt
mw

where

grd1 : MapSplit = TRUE
grd2 : mt ∈ dom(msplit)
grd3 : mw ∈ MW
grd4 : mt ↦ mw ∉ ExecMT
grd6 : mw ∉ dom(mworker_result)
grd7 : mt ↦ mw ∈ AssignMT

then

act1 : ExecMT := ExecMT ∪ {mt ↦ mw}
act2 : mworker_result := mworker_result∪{mw ↦ {mt ↦ map(msplit(mt))}}

end

Event *MTSuccess* ≙

extends *MTSuccess*

any

mt
mw

where

grd1 : mt ∈ dom(msplit)
grd2 : mt ∉ MTSuccess
grd3 : mt ↦ mw ∈ ExecMT

then

act1 : MTSuccess := MTSuccess ∪ {mt}
act2 : MTProcess_result(mt) := mworker_result(mw)(mt)

end

Event *MapCommit* ≙

extends *MapCommit*

when

grd1 : MapSplit = TRUE
grd2 : MapCommit = FALSE
grd3 : MTSuccess = dom(msplit)

then

act1 : MapCommit := TRUE
act2 : map_result := (⋃ mt·mt ∈ dom(msplit)|{mt}×MTProcess_result(mt))

end

Event *ReduceSplit* ≙

extends *ReduceSplit*

when

grd1 : MapCommit = TRUE
grd2 : ReduceSplit = FALSE

then

act1 : ReduceSplit := TRUE
act2 : rsplit := RSplit(map_result)

end

Event *AssignRT* ≙

any

rt

rw

where

grd1 : $ReduceSplit = TRUE$

grd2 : $rt \in dom(rsplit)$

grd3 : $rw \in RW$

grd4 : $rt \mapsto rw \notin AssignRT$

grd5 : $rw \notin ran(AssignRT)$

then

act1 : $AssignRT := AssignRT \cup \{rt \mapsto rw\}$

end

Event $ExecRT \widehat{=}$

refines $RTok$

any

rt

rw

where

grd1 : $ReduceSplit = TRUE$

grd2 : $rt \in dom(rsplit)$

grd3 : $rw \in RW$

grd4 : $rt \mapsto rw \notin ExecRT$

grd5 : $rw \notin dom(rworker_result)$

grd6 : $rt \mapsto rw \in AssignRT$

then

act1 : $ExecRT := ExecRT \cup \{rt \mapsto rw\}$

act2 : $rworker_result := rworker_result \cup \{rw \mapsto \{rt \mapsto reduce(rsplit(rt))\}\}$

end

Event $RTSuccess \widehat{=}$

extends $RTSuccess$

any

rt

rw

where

grd1 : ReduceSplit = TRUE

grd2 : rt ∈ dom(rsplit)

grd3 : rt ∉ RTSuccess

grd4 : rt ↦ rw ∈ ExecRT

then

act1 : RTSuccess := RTSuccess ∪ {rt}

act2 : RTProcess_result(rt) := rworker_result(rw)(rt)

end

Event $ReduceCommit \widehat{=}$

extends $ReduceCommit$

when

grd1 : $\texttt{ReduceSplit} = \texttt{TRUE}$

grd2 : $\texttt{ReduceCommit} = \texttt{FALSE}$

grd3 : $\texttt{RTSuccess} = \texttt{dom}(\texttt{rsplit})$

then

act1 : $\texttt{ReduceCommit} := \texttt{TRUE}$

act2 : $\texttt{reduce_result} := (\bigcup \texttt{rt} \cdot \texttt{rt} \in \texttt{dom}(\texttt{rsplit}) | \texttt{RTProcess_result}(\texttt{rt}))$

end

Event *OutputMapReduce* $\widehat{=}$

extends *OutputMapReduce*

when

grd1 : $\texttt{ReduceCommit} = \texttt{TRUE}$

grd2 : $\texttt{OutputMapReduce} = \texttt{FALSE}$

then

act1 : $\texttt{output} := \texttt{Combine}(\texttt{reduce_result})$

act2 : $\texttt{OutputMapReduce} := \texttt{TRUE}$

end

END

2. MapReduce: Partially Blocking Model

CONTEXT MapReduce2_cnt

SETS

`IData` the input data to be processed

`OData` the resulting output data

`MTask` a set of all possible map tasks

`MData` an abstract type defining the data obtained after map splitting

`RTask` a set of all possible reduce tasks

`RData` an abstract type defining the intermediate data obtained during the Map phase

`MW` a set of map workers

`RW` a set of reduce workers

CONSTANTS

`MSplit`

`Map`

`RSplit`

`Reduce`

`Combine`

`mnum`

`rnum`

`map` the user-supplied function map

`reduce` the user-supplied function reduce

`dep` the dependency relation between map and reduce tasks

`rsplit`

`mtasks`

`rtasks`

AXIOMS

`axm1` : $finite(IData)$

`axm2` : $finite(MTask)$

`axm3` : $finite(MData)$

`axm4` : $finite(RData)$

`axm5` : $finite(RTask)$

`axm6` : $finite(MW)$

`axm7` : $finite(RW)$

`axm8` : $MSplit \in IData \rightarrow (MTask \nrightarrow MData)$

`axm9` : $Map \in (MTask \nrightarrow MData) \rightarrow \mathbb{P}_1(MTask \times RData)$

`axm10` : $RSplit \in \mathbb{P}_1(MTask \times RData) \rightarrow (RTask \nrightarrow \mathbb{P}_1(RData))$

`axm11` : $Reduce \in (RTask \nrightarrow \mathbb{P}_1(RData)) \rightarrow \mathbb{P}_1(OData)$

`axm12` : $Combine \in \mathbb{P}_1(OData) \rightarrow OData$

axm13 : $\forall f \cdot f \in RTask \pfun \mathbb{P}_1(RData) \Rightarrow finite(f)$

axm14 : $\forall f \cdot f \in MTask \pfun MData \Rightarrow finite(f)$

axm15 : $\forall f \cdot f \in MTask \pfun MData \Rightarrow dom(f) = dom(Map(f))$

axm16 : $\forall f \cdot f \in \mathbb{P}_1(MTask \times RData) \Rightarrow ran(f) = (\bigcup rt \cdot rt \in dom(RSplit(f)) | RSplit(f)(rt))$

axm17 : $mtasks \in IData \tfun \mathbb{P}_1(MTask)$

axm18 : $mnum \in IData \tfun \mathbb{N}_1$

axm19 : $\forall idata \cdot idata \in IData \Rightarrow card(MSplit(idata)) = mnum(idata)$

axm20 : $\forall idata \cdot idata \in IData \Rightarrow mtasks(idata) = dom(MSplit(idata))$

axm21 : $\forall idata \cdot idata \in IData \Rightarrow card(mtasks(idata)) = mnum(idata)$

axm22 : $rtasks \in IData \tfun \mathbb{P}_1(RTask)$

axm23 : $rnum \in IData \tfun \mathbb{N}_1$

axm24 : $\forall idata \cdot idata \in IData \Rightarrow card(RSplit(Map(MSplit(idata)))) = rnum(idata)$

axm25 : $\forall idata \cdot idata \in IData \Rightarrow rtasks(idata) = dom(RSplit(Map(MSplit(idata))))$

axm26 : $\forall idata \cdot idata \in IData \Rightarrow card(rtasks(idata)) = rnum(idata)$

axm27 : $map \in MData \tfun \mathbb{P}_1(RData)$

axm28 : $\forall idata, f \cdot idata \in IData \land f \subseteq MSplit(idata) \Rightarrow Map(f) = (\bigcup mt \cdot mt \in dom(f) | \{mt\} \times map(f(mt)))$

axm29 : $reduce \in \mathbb{P}_1(RData) \tfun \mathbb{P}_1(OData)$

axm30 : $\forall idata, f \cdot idata \in IData \land f \subseteq RSplit(Map(MSplit(idata))) \Rightarrow Reduce(f) = (\bigcup rt \cdot rt \in dom(f) | reduce(f(rt)))$

axm31 : $rsplit \in RTask \pfun (\mathbb{P}_1(MTask \times RData) \tfun \mathbb{P}_1(RData))$

axm32 : $\forall f \cdot f \in ran(Map) \Rightarrow RSplit(f) = (\bigcup rt \cdot rt \in dom(rsplit) | \{rt \mapsto rsplit(rt)(f)\})$

axm33 : $\forall idata \cdot idata \in IData \Rightarrow dom(RSplit(Map(MSplit(idata)))) = dom(rsplit)$

axm34 : $\forall idata \cdot idata \in IData \Rightarrow rtasks(idata) = dom(rsplit)$

axm35 : $dep \in IData \tfun \mathbb{P}(RTask \times MTask)$

axm36 : $\forall idata, rt, mt \cdot idata \in IData \land rt \in RTask \land mt \in MTasks \land rt \mapsto mt \in dep(idata) \Leftrightarrow$
$mt \in dom(MSplit(idata)) \land$
$(\exists rd \cdot rd \in RData \land rt \in dom(RSplit(Map(MSplit(idata)))) \land$
$rd \in RSplit(Map(MSplit(idata)))(rt) \land mt \mapsto rd \in Map(MSplit(idata)))$

axm37 : $\forall idata \cdot idata \in IData \Rightarrow dom(dep(idata)) = rtasks(idata)$

axm38 : $\forall idata \cdot idata \in IData \Rightarrow ran(dep(idata)) = mtasks(idata)$

axm39 : $\forall idata \cdot idata \in IData \Rightarrow (\forall rt \cdot rt \in rtasks(idata) \Rightarrow Map((dep(idata))[\{rt\}] \domrest MSplit(idata)) = \varnothing)$

axm40 : $\forall idata \cdot idata \in IData \Rightarrow (\forall f, rt \cdot f = Map(MSplit(idata)) \land rt \in rtasks(idata) \Rightarrow$
$rsplit(rt)(f) = RSplit(f)(rt))$

END

MACHINE MapReduce2_m0

SEES MapReduce2_cnt

VARIABLES

output

idata

done

INVARIANTS

inv1 : $output \in OData$

inv2 : $idata \in IData$

inv3 : $done \in BOOL$

inv4 : $done = TRUE \Rightarrow output = Combine(Reduce(RSplit(Map(MSplit(idata)))))$

EVENTS

Initialisation

begin

act1 : $output :\in OData$

act2 : $idata :\in IData$

act3 : $done := FALSE$

end

Event $OutputMapReduce \mathrel{\widehat{=}}$

any

$t1$

$t2$

$t3$

$t4$

$pmtask$

$prtask$

where

grd1 : $pmtask \subseteq MTask$

grd2 : $prtask \subseteq RTask$

grd3 : $t1 \in pmtask \rightarrow MData$

grd4 : $t2 \in \mathbb{P}_1(pmtask \times RData)$

grd5 : $t3 \in prtask \rightarrow \mathbb{P}_1(RData)$

grd6 : $t4 \in \mathbb{P}_1(OData)$

grd7 : $t1 = MSplit(idata)$

grd8 : $t2 = Map(t1)$

grd9 : $t3 = RSplit(t2)$

grd10 : $t4 = Reduce(t3)$

then

act1 : $output := Combine(t4)$

act2 : $done := TRUE$

end

END

MACHINE MapReduce2_m1

REFINES MapReduce2_m0

SEES MapReduce2_cnt

VARIABLES

`output`

`idata`

`MapSplit`

`OutputMapReduce`

`msplit`

INVARIANTS

`inv1` : $MapSplit \in BOOL$

`inv2` : $OutputMapReduce \in BOOL$

`inv3` : $OutputMapReduce = done$

`inv4` : $OutputMapReduce = TRUE \Rightarrow MapSplit = TRUE$

`inv5` : $msplit \in MTask \pfun MData$

`inv6` : $MapSplit = TRUE \Rightarrow msplit = MSplit(idata)$

`inv7` : $MapSplit = TRUE \Rightarrow card(msplit) = mnum(idata)$

`inv8` : $MapSplit = FALSE \Rightarrow msplit = \varnothing$

`inv9` : $MapSplit = TRUE \Rightarrow Map(msplit) \in \mathbb{P}_1(dom(msplit) \times RData)$

`inv10` : $MapSplit = TRUE \Rightarrow dom(msplit) = mtasks(idata)$

EVENTS

Initialisation

begin

`act1` : $output :\in OData$

`act2` : $idata :\in IData$

`act3` : $MapSplit := FALSE$

`act4` : $OutputMapReduce := FALSE$

`act5` : $msplit := \varnothing$

end

Event $MapSplit \mathrel{\widehat{=}}$

when

`grd1` : $MapSplit = FALSE$

then

`act1` : $MapSplit := TRUE$

`act2` : $msplit := MSplit(idata)$

end

Event $OutputMapReduce \mathrel{\widehat{=}}$

refines $OutputMapReduce$

any

$t2$

$t3$

$t4$
$prtask$

where

grd1 : $MapSplit = TRUE$
grd2 : $OutputMapReduce = FALSE$
grd3 : $prtask \subseteq RTask$
grd4 : $t2 \in \mathbb{P}_1(dom(msplit) \times RData)$
grd5 : $t3 \in prtask \rightarrow \mathbb{P}_1(RData)$
grd6 : $t4 \in \mathbb{P}_1(OData)$
grd7 : $t2 = Map(msplit)$
grd8 : $t3 = RSplit(t2)$
grd9 : $t4 = Reduce(t3)$

with

pmtask : $\texttt{pmtask} = \texttt{dom}(\texttt{msplit})$
t1 : $\texttt{t1} = \texttt{msplit}$

then

act1 : $output := Combine(t4)$
act2 : $OutputMapReduce := TRUE$

end

END

MACHINE MapReduce2_m2

REFINES MapReduce2_m1

SEES MapReduce2_cnt

VARIABLES

`MapSplit`

`OutputMapReduce`

`idata`

`msplit`

`output`

`RSplitReducePhase`

`rsplit_result`

`reduce_result`

INVARIANTS

`inv1` : $RSplitReducePhase \in BOOL$

`inv2` : $OutputMapReduce = TRUE \Rightarrow RSplitReducePhase = TRUE$

`inv3` : $RSplitReducePhase = TRUE \Rightarrow MapSplit = TRUE$

`inv4` : $rsplit_result \in RTask \pfun \mathbb{P}_1(RData)$

`inv5` : $reduce_result \in \mathbb{P}(OData)$

`inv6` : $RSplitReducePhase = FALSE \Rightarrow rsplit_result = \varnothing \land reduce_result = \varnothing$

`inv7` : $RSplitReducePhase = TRUE \Rightarrow card(rsplit_result) = rnum(idata)$

`inv8` : $RSplitReducePhase = TRUE \Rightarrow rsplit_result = RSplit(Map(msplit)) \land reduce_result = Reduce(RSplit(Map(msplit)))$

EVENTS

Initialisation

extended

begin

`act1` : `output` $:\in$ `OData`

`act2` : `idata` $:\in$ `IData`

`act3` : `MapSplit` $:=$ `FALSE`

`act4` : `OutputMapReduce` $:=$ `FALSE`

`act5` : `msplit` $:= \varnothing$

`act6` : $RSplitReducePhase := FALSE$

`act7` : $rsplit_result := \varnothing$

`act8` : $reduce_result := \varnothing$

end

Event $MapSplit \mathrel{\widehat{=}}$

extends *MapSplit*

when

`grd1` : `MapSplit` $=$ `FALSE`

then

`act1` : `MapSplit` $:=$ `TRUE`

act2 : msplit := MSplit(idata)

end

Event *RSplitReducePhase* $\widehat{=}$

any

p1

p2

where

grd1 : $MapSplit = TRUE$

grd2 : $RSplitReducePhase = FALSE$

grd3 : $p1 = RSplit(Map(MSplit(idata)))$

grd4 : $p2 = Reduce(RSplit(Map(MSplit(idata))))$

then

act1 : $RSplitReducePhase := TRUE$

act2 : $rsplit_result := p1$

act3 : $reduce_result := p2$

end

Event *OutputMapReduce* $\widehat{=}$

refines *OutputMapReduce*

any

t2

where

grd1 : $RSplitReducePhase = TRUE$

grd2 : $OutputMapReduce = FALSE$

grd3 : $t2 \in \mathbb{P}_1(dom(msplit) \times RData)$

grd5 : $t2 = Map(msplit)$

with

t3 : t3 = rsplit_result

prtask : prtask = dom(rsplit_result)

t4 : t4 = reduce_result

then

act1 : $output := Combine(reduce_result)$

act2 : $OutputMapReduce := TRUE$

end

END

MACHINE MapReduce2_m3

REFINES MapReduce2_m2

SEES MapReduce2_cnt

VARIABLES

`MapSplit`

`OutputMapReduce`

`idata`

`msplit`

`output`

`RSplitReduceCommit`

`rsplit_result`

`reduce_result`

`RTSplitReduceProcess`

INVARIANTS

`inv1` : $RSplitReduceCommit \in BOOL$

`inv2` : $RSplitReduceCommit = RSplitReducePhase$

`inv3` : $RTSplitReduceProcess \subseteq rtasks(idata)$

`inv4` : $RSplitReduceCommit = TRUE \Rightarrow MapSplit = TRUE$

`inv5` : $OutputMapReduce = TRUE \Rightarrow RSplitReduceCommit = TRUE$

`inv6` : $RTSplitReduceProcess \neq \varnothing \Rightarrow MapSplit = TRUE$

`inv7` : $RSplitReduceCommit = TRUE \Rightarrow RTSplitReduceProcess = rtasks(idata)$

EVENTS

Initialisation

begin

`act1` : $output :\in OData$

`act2` : $idata :\in IData$

`act3` : $MapSplit := FALSE$

`act4` : $OutputMapReduce := FALSE$

`act5` : $msplit := \varnothing$

`act6` : $RSplitReduceCommit := FALSE$

`act7` : $rsplit_result := \varnothing$

`act8` : $reduce_result := \varnothing$

`act9` : $RSTplitReduceProcess := \varnothing$

end

Event $MapSplit \mathrel{\widehat{=}}$

extends $MapSplit$

when

`grd1` : `MapSplit` = `FALSE`

then

`act1` : `MapSplit` := `TRUE`

`act2` : `msplit` := `MSplit(idata)`

end

Event $RTSplitReduceProcess \widehat{=}$

any

rt

where

grd1 : $MapSplit = TRUE$

grd2 : $rt \in rtasks(idata)$

grd3 : $rt \notin RTSplitReduceProcess$

then

act1 : $RTSplitReduceProcess := RTSplitReduceProcess \cup \{rt\}$

end

Event $RSplitReduceCommit \widehat{=}$

refines $RSplitReducePhase$

any

$p1$

$p2$

where

grd1 : $RTSplitReduceProcess = rtasks(idata)$

grd2 : $RSplitReduceCommit = FALSE$

grd3 : $p1 = RSplit(Map(MSplit(idata)))$

grd4 : $p2 = Reduce(RSplit(Map(MSplit(idata))))$

then

act1 : $RSplitReduceCommit := TRUE$

act2 : $rsplit_result := p1$

act3 : $reduce_result := p2$

end

Event $OutputMapReduce \widehat{=}$

refines $OutputMapReduce$

any

$t2$

where

grd1 : $RSplitReduceCommit = TRUE$

grd2 : $OutputMapReduce = FALSE$

grd3 : $t2 \in \mathbb{P}_1(dom(msplit) \times RData)$

grd5 : $t2 = Map(msplit)$

then

act1 : $output := Combine(reduce_result)$

act2 : $OutputMapReduce := TRUE$

end

END

MACHINE MapReduce2_m4

REFINES MapReduce2_m3

SEES MapReduce2_cnt

VARIABLES

`MapSplit`

`OutputMapReduce`

`idata`

`msplit`

`output`

`RSplitReduceCommit`

`rsplit_result`

`reduce_result`

`RTProcess`

`RTSplitProcess`

`RTSplit_result`

`Reduce_result`

INVARIANTS

inv1 : $RTSplitProcess \subseteq rtasks(idata)$

inv2 : $RTSplitReduceProcess = RTProcess$

inv3 : $RTProcess \subseteq RTSplitProcess$

inv4 : $RTSplit_result \in rtasks(idata) \pfun \mathbb{P}_1(RData)$

inv5 : $Reduce_result \in rtasks(idata) \pfun \mathbb{P}_1(OData)$

inv6 : $MapSplit = FALSE \Rightarrow RTSplitProcess = \varnothing$

inv7 : $dom(RTSplit_result) = RTSplitProcess$

inv8 : $dom(Reduce_result) = RTProcess$

inv9 : $\forall rt \cdot rt \in dom(RTSplit_result) \Rightarrow RTSplit_result(rt) = rsplit(rt)(Map(dep(idata)[\{rt\}] \lhd MSplit(idata)))$

inv10 : $RSplitReduceCommit = TRUE \Rightarrow RTSplitProcess = rtasks(idata)$

inv11 : $\forall rt \cdot rt \in dom(Reduce_result) \Rightarrow Reduce_result(rt) = reduce(RTSplit_result(rt))$

inv12 : $\forall rt \cdot rt \in RTSplitProcess \Rightarrow dep(idata)[\{rt\}] \subseteq dom(MSplit(idata))$

EVENTS

Initialisation

begin

act1 : $output :\in OData$

act2 : $idata :\in IData$

act3 : $MapSplit := FALSE$

act4 : $OutputMapReduce := FALSE$

act5 : $msplit := \varnothing$

act6 : $RSplitReduceCommit := FALSE$

act7 : $rsplit_result := \varnothing$

act8 : $reduce_result := \varnothing$
act9 : $RTProcess := \varnothing$
act10 : $RTSplitProcess := \varnothing$
act12 : $Reduce_result := \varnothing$
act13 : $RTSplit_result := \varnothing$

end

Event $MapSplit \mathrel{\widehat{=}}$

extends $MapSplit$

when

grd1 : $\texttt{MapSplit} = \texttt{FALSE}$

then

act1 : $\texttt{MapSplit} := \texttt{TRUE}$
act2 : $\texttt{msplit} := \texttt{MSplit}(\texttt{idata})$

end

Event $RTSplitProcess \mathrel{\widehat{=}}$

any

rt
$presult$
mts

where

grd1 : $MapSplit = TRUE$
grd2 : $rt \in rtasks(idata)$
grd3 : $rt \notin RTSplitProcess$
grd4 : $mts = dep(idata)[\{rt\}]$
grd5 : $mts \subseteq dom(msplit)$
grd6 : $presult = Map(mts \lhd MSplit(idata))$

then

act1 : $RTSplitProcess := RTSplitProcess \cup \{rt\}$
act2 : $RTSplit_result(rt) := rsplit(rt)(presult)$

end

Event $RTProcess \mathrel{\widehat{=}}$

refines $RTSplitReduceProcess$

any

rt

where

grd1 : $rt \notin RTProcess$
grd2 : $rt \in RTSplitProcess$

then

act1 : $RTProcess := RTProcess \cup \{rt\}$
act2 : $Reduce_result(rt) := reduce(RTSplit_result(rt))$

end

Event $RSplitReduceCommit \mathrel{\widehat{=}}$

refines $RSplitReduceCommit$

when

grd1 : $RTProcess = rtasks(idata)$

grd2 : $RSplitReduceCommit = FALSE$

with

p1 : $\texttt{p1} = \texttt{RTSplit_result}$

p2 : $\texttt{p2} = (\bigcup \texttt{rt} \cdot \texttt{rt} \in \texttt{rtasks}(\texttt{idata}) | \texttt{Reduce_result}(\texttt{rt}))$

then

act1 : $RSplitReduceCommit := TRUE$

act2 : $rsplit_result := RSplit_result$

act3 : $reduce_result := (\bigcup rt \cdot rt \in rtasks(idata) | Reduce_result(rt))$

end

Event $OutputMapReduce \mathrel{\widehat{=}}$

extends $OutputMapReduce$

any

t2

where

grd1 : $\texttt{RSplitReduceCommit} = \texttt{TRUE}$

grd2 : $\texttt{OutputMapReduce} = \texttt{FALSE}$

grd3 : $\texttt{t2} \in \mathbb{P}_1(\texttt{dom}(\texttt{msplit}) \times \texttt{RData})$

grd5 : $\texttt{t2} = \texttt{Map}(\texttt{msplit})$

then

act1 : $\texttt{output} := \texttt{Combine}(\texttt{reduce_result})$

act2 : $\texttt{OutputMapReduce} := \texttt{TRUE}$

end

END

MACHINE MapReduce2_m5

REFINES MapReduce2_m4

SEES MapReduce2_cnt

VARIABLES

`MapSplit`

`OutputMapReduce`

`idata`

`msplit`

`output`

`RSplitReduceCommit`

`rsplit_result`

`reduce_result`

`RTProcess`

`RTSplitProcess`

`RTSplit_result`

`Reduce_result`

`MTProcess_result`

`MTProcess`

INVARIANTS

`inv1` : $MTProcess \subseteq RTask \times MTask$

`inv2` : $MTProcess \neq \varnothing \Rightarrow MapSplit = TRUE$

`inv3` : $MTProcess_result \in MTask \nrightarrow \mathbb{P}_1(RData)$

`inv4` : $\forall mt \cdot mt \in ran(MTProcess) \Rightarrow mt \in dom(MTProcess_result)$

`inv5` : $RTSplit = RTSplitProcess$

`inv6` : $RTSplit \subseteq dom(MTProcess)$

`inv7` : $RTSplit\varnothing \Rightarrow MapSplit = TRUE$

`inv8` : $dom(MTProcess_result) \subseteq mtasks(idata)$

`inv9` : $dom(MTProcess_result) \subseteq dom(msplit)$

`inv10` : $dom(MTProcess_result) = ran(MTProcess)$

`inv11` : $\forall mt \cdot mt \in dom(MTProcess_result) \Rightarrow MTProcess_result(mt) = map(msplit(mt))$

`inv12` : $RTSplit \neq \varnothing \Rightarrow MapSplit = TRUE$

`inv13` : $dep(idata)[RSplit] \subseteq ran(MTProcess)$

`inv14` : $\forall rt \cdot rt \in RTSplit \Rightarrow dep(idata)[\{rt\}] \subseteq ran(MTProcess)$

EVENTS

Initialisation

extended

begin

`act1` : `output` :∈ `OData`

`act2` : `idata` :∈ `IData`

act3 : MapSplit := FALSE
act4 : OutputMapReduce := FALSE
act5 : msplit := ∅
act6 : RSplitReduceCommit := FALSE
act7 : rsplit_result := ∅
act8 : reduce_result := ∅
act9 : RTProcess := ∅
act10 : RTSplit := ∅
act11 : Reduce_result := ∅
act12 : RSplit_result := ∅
act13 : $MTProcess := \varnothing$
act14 : $MTProcess_result := \varnothing$

end

Event $MapSplit \mathrel{\widehat{=}}$

extends $MapSplit$

when

grd1 : MapSplit = FALSE

then

act1 : MapSplit := TRUE
act2 : msplit := MSplit(idata)

end

Event $MTProcess \mathrel{\widehat{=}}$

any

mts
mt
rts
rt

where

grd1 : $MapSplit = TRUE$
grd2 : $rt \in rtasks(idata)$
grd3 : $mts = dep2(idata)[\{rt\}]$
grd4 : $mt \in mts$
grd5 : $rt \notin dom(MTProcess)$
grd6 : $mt \notin ran(MTProcess)$
grd7 : $rts = dep(idata)^{-1}[\{mt\}]$
grd8 : $mts \subseteq mtasks(idata)$

then

act1 : $MTProcess := MTProcess \cup (rts \times \{mt\})$
act2 : $MTProcess_result(mt) := map(msplit(mt))$

end

Event $RTSplit \mathrel{\widehat{=}}$

refines $RSplitProcess$

any

rt

mts

$presult$

where

grd2 : $rt \in rtasks(idata)$

grd3 : $rt \notin RTSplit$

grd4 : $mts \subseteq ran(MTProcess)$

grd5 : $mts = dep(idata)[\{rt\}]$

grd6 : $presult = (\bigcup mt \cdot mt \in mts | \{mt\} \times MTProcess_result(mt))$

grd7 : $rt \in dom(MTProcess)$

then

act1 : $RTSplit := RTSplit \cup \{rt\}$

act2 : $RTSplit_result(rt) := rsplit(rt)(presult)$

end

Event $RTProcess \mathrel{\widehat{=}}$

extends $RTProcess$

any

rt

where

grd1 : $\mathtt{rt} \notin \mathtt{RTProcess}$

grd2 : $\mathtt{rt} \in \mathtt{RTSplit}$

then

act1 : $\mathtt{RTProcess} := \mathtt{RTProcess} \cup \{\mathtt{rt}\}$

act2 : $\mathtt{Reduce_result}(\mathtt{rt}) := \mathtt{reduce}(\mathtt{RTSplit_result}(\mathtt{rt}))$

end

Event $RSplitReduceCommit \mathrel{\widehat{=}}$

extends $RSplitReduceCommit$

when

grd1 : $\mathtt{RTProcess} = \mathtt{rtasks}(\mathtt{idata})$

grd2 : $\mathtt{RSplitReduceCommit} = \mathtt{FALSE}$

then

act1 : $\mathtt{RSplitReduceCommit} := \mathtt{TRUE}$

act2 : $\mathtt{rsplit_result} := \mathtt{RTSplit_result}$

act3 : $\mathtt{reduce_result} := (\bigcup \mathtt{rt} \cdot \mathtt{rt} \in \mathtt{rtasks}(\mathtt{idata}) | \mathtt{Reduce_result}(\mathtt{rt}))$

end

Event $OutputMapReduce \mathrel{\widehat{=}}$

refines $OutputMapReduce$

when

grd1 : $RSplitReduceCommit = TRUE$

grd2 : $OutputMapReduce = FALSE$

with

t2 : $\mathtt{t2} = (\bigcup \mathtt{mt} \cdot \mathtt{mt} \in \mathtt{dom}(\mathtt{MapProcess_result}) | \{\mathtt{mt}\} \times \mathtt{MTProcess_result}(\mathtt{mt}))$

then

act1 : $output := Combine(reduce_result)$

act2 : $OutputMapReduce := TRUE$

```
        end
END
```

MACHINE MapReduce2_m6

REFINES MapReduce2_m5

SEES MapReduce2_cnt

VARIABLES

`MapSplit`

`OutputMapReduce`

`idata`

`msplit`

`output`

`RSplitReduceCommit`

`rsplit_result`

`reduce_result`

`ReduceProcess`

`RSplitProcess`

`RSplit_result`

`Reduce_result`

`MTProcess_result`

`MTok`

`MTSuccess`

`mworker_result`

INVARIANTS

`inv1` : $MTok \subseteq RTask \times MTask \times MW$

`inv2` : $MTSuccess \subseteq RTask \times MTask$

`inv3` : $MTSuccess = MTProcess$

`inv4` : $MTok \neq \varnothing \Rightarrow MapSplit = TRUE$

`inv5` : $mworker_result \in MW \pfun (MTask \pfun \mathbb{P}_1(RData))$

`inv6` : $MapSplit = FALSE \Rightarrow dom(mworker_result) = \varnothing$

`inv7` : $ran(MTSuccess) \subseteq ran(dom(MTok))$

`inv8` : $dom(mworker_result) = ran(MTok)$

`inv9` : $\forall rt, mt, mw \cdot rt \mapsto mt \mapsto mw \in MTok \Rightarrow mw \in dom(mworker_result) \land mt \in dom(mworker_result(mw))$

`inv10` : $ran(dom(MTok)) \subseteq mtasks(idata)$

`inv11` : $\forall rt, mt, mw \cdot rt \mapsto mt \mapsto mw \in MTok \Rightarrow mworker_result(mw)(mt) = map(msplit(mt))$

EVENTS

Initialisation

begin

`act1` : $output :\in OData$

`act2` : $idata :\in IData$

`act3` : $MapSplit := FALSE$

act4 : $OutputMapReduce := FALSE$
act5 : $msplit := \varnothing$
act6 : $RSplitReduceCommit := FALSE$
act7 : $rsplit_result := \varnothing$
act8 : $reduce_result := \varnothing$
act9 : $RTProcess := \varnothing$
act10 : $RTSplit := \varnothing$
act12 : $Reduce_result := \varnothing$
act13 : $RTSplit_result := \varnothing$
act14 : $MTSuccess := \varnothing$
act15 : $MTProcess_result := \varnothing$
act16 : $MTok := \varnothing$
act17 : $mworker_result := \varnothing$

end

Event *MapSplit* $\widehat{=}$

extends *MapSplit*

when

grd1 : MapSplit = FALSE

then

act1 : MapSplit := TRUE
act2 : msplit := MSplit(idata)

end

Event *MTok* $\widehat{=}$

any

mt
rt
mw

where

grd1 : $MapSplit = TRUE$
grd2 : $mt \in mtasks(idata)$
grd3 : $rt \in rtasks(idata)$
grd4 : $mw \in MW$
grd5 : $mw \notin dom(mworker_result)$
grd6 : $mt \notin ran(dom(MTok))$

then

act1 : $MTok := MTok \cup \{rt \mapsto mt \mapsto mw\}$
act2 : $mworker_result := mworker_result \cup \{mw \mapsto \{mt \mapsto map(msplit(mt))\}\}$

end

Event *MTSuccess* $\widehat{=}$

refines *MTProcess*

any

mts
mt
rts

rt

mw

where

grd1 : $MapSplit = TRUE$

grd2 : $rt \in rtasks(idata)$

grd3 : $mts = dep(idata)[\{rt\}]$

grd4 : $mt \in mts$

grd5 : $rt \notin dom(MTSuccess)$

grd6 : $mt \notin ran(MTSuccess)$

grd7 : $rts = dep(idata)^{-1}[\{mt\}]$

grd8 : $mts \subseteq mtasks(idata)$

grd9 : $\exists rt1 \cdot rt1 \in rts \land (rt1 \mapsto mt \mapsto mw) \in MTok$

then

act1 : $MTSuccess := MTSuccess \cup (rts \times \{mt\})$

act2 : $MTProcess_result(mt) := mworker_result(mw)(mt)$

end

Event $RTSplit \mathrel{\widehat{=}}$

refines $RTSplit$

any

rt

mts

$presult$

where

grd2 : $rt \in rtasks(idata)$

grd3 : $rt \notin RTSplit$

grd4 : $mts \subseteq ran(MTSuccess)$

grd5 : $mts = dep(idata)[\{rt\}]$

grd6 : $presult = (\bigcup mt \cdot mt \in mts | \{mt\} \times MTProcess_result(mt))$

grd7 : $rt \in dom(MTSuccess)$

then

act1 : $RTSplit := RTSplit \cup \{rt\}$

act2 : $RTSplit_result(rt) := rsplit(rt)(presult)$

end

Event $RTProcess \mathrel{\widehat{=}}$

extends $RTProcess$

any

`rt`

where

grd1 : $\texttt{rt} \notin \texttt{RTProcess}$

grd2 : $\texttt{rt} \in \texttt{RTSplitProcess}$

then

act1 : $\texttt{RTProcess} := \texttt{RTProcess} \cup \{\texttt{rt}\}$

act2 : $\texttt{Reduce_result}(\texttt{rt}) := \texttt{reduce}(\texttt{RTSplit_result}(\texttt{rt}))$

end

Event *RSplitReduceCommit* $\widehat{=}$
extends *RSplitReduceCommit*
when
grd1 : $\texttt{RTProcess} = \texttt{rtasks}(\texttt{idata})$
grd2 : $\texttt{RSplitReduceCommit} = \texttt{FALSE}$
then
act1 : $\texttt{RSplitReduceCommit} := \texttt{TRUE}$
act2 : $\texttt{rsplit_result} := \texttt{RSplit_result}$
act3 : $\texttt{reduce_result} := (\bigcup \texttt{rt} \cdot \texttt{rt} \in \texttt{rtasks}(\texttt{idata}) | \texttt{Reduce_result}(\texttt{rt}))$
end
Event *OutputMapReduce* $\widehat{=}$
extends *OutputMapReduce*
when
grd1 : $\texttt{RSplitReduceCommit} = \texttt{TRUE}$
grd2 : $\texttt{OutputMapReduce} = \texttt{FALSE}$
then
act1 : $\texttt{output} := \texttt{Combine}(\texttt{reduce_result})$
act2 : $\texttt{OutputMapReduce} := \texttt{TRUE}$
end
END

MACHINE MapReduce2_m7

REFINES MapReduce2_m6

SEES MapReduce2_cnt

VARIABLES

`MapSplit`
`OutputMapReduce`
`idata`
`msplit`
`output`
`RSplitReduceCommit`
`rsplit_result`
`reduce_result`
`RTProcess`
`RTSplit`
`RTSplit_result`
`Reduce_result`
`MapProcess_result`
`ExecMT`
`MTSuccess`
`mworker_result`
`AssignMT`

INVARIANTS

`inv1` : $AssignMT \subseteq RTask \times MTask \times MW$
`inv2` : $ExecMT = MTok$
`inv3` : $ran(dom(ExecMT)) \subseteq ran(dom(AssignMT))$
`inv4` : $AssignMT \neq \varnothing \Rightarrow MapSplit = TRUE$
`inv5` : $dom(dom(AssignMT)) \subseteq rtasks(idata)$

EVENTS

Initialisation

begin

`act1` : $output :\in OData$
`act2` : $idata :\in IData$
`act3` : $MapSplit := FALSE$
`act4` : $OutputMapReduce := FALSE$
`act5` : $msplit := \varnothing$
`act6` : $RSplitReduceCommit := FALSE$
`act7` : $rsplit_result := \varnothing$
`act8` : $reduce_result := \varnothing$
`act9` : $ReduceProcess := \varnothing$
`act10` : $RSplitProcess := \varnothing$
`act12` : $Reduce_result := \varnothing$

act13 : $RSplit_result := \varnothing$
act14 : $MTSuccess := \varnothing$
act15 : $MTProcess_result := \varnothing$
act16 : $ExecMT := \varnothing$
act17 : $mworker_result := \varnothing$
act18 : $AssignMT := \varnothing$

end

Event $MapSplit \mathrel{\widehat{=}}$

extends $MapSplit$

when

grd1 : MapSplit = FALSE

then

act1 : MapSplit := TRUE
act2 : msplit := MSplit(idata)

end

Event $AssignMT \mathrel{\widehat{=}}$

any

rt
mt
mw

where

grd1 : $MapSplit = TRUE$
grd2 : $rt \in rtasks(idata)$
grd3 : $mt \in mtasks(idata)$
grd4 : $mw \in MW$
grd5 : $rt \mapsto mt \mapsto mw \notin AssignMT$
grd6 : $mw \notin ran(AssignMT)$

then

act1 : $AssignMT := AssignMT \cup \{rt \mapsto mt \mapsto mw\}$

end

Event $ExecMT \mathrel{\widehat{=}}$

refines $MTok$

any

mt
rt
mw

where

grd2 : $mt \in mtasks(idata)$
grd3 : $rt \in rtasks(idata)$
grd5 : $mw \notin dom(mworker_result)$
grd6 : $mt \notin ran(dom(ExecMT))$
grd7 : $\exists rt1 \cdot rt1 \in rtasks(idata) \land rt1 \mapsto mt \mapsto mw \in AssignMT$

then

act1 : $ExecMT := ExecMT \cup \{rt \mapsto mt \mapsto mw\}$

act2 : $mworker_result := mworker_result \cup \{mw \mapsto \{mt \mapsto map(msplit(mt))\}\}$

end

Event $MTSuccess \mathrel{\widehat{=}}$

refines $MTSuccess$

any

mts
mt
rts
rt
mw

where

grd1 : $MapSplit = TRUE$
grd2 : $rt \in rtasks(idata)$
grd3 : $mts = dep(idata)[\{rt\}]$
grd4 : $mt \in mts$
grd5 : $rt \notin dom(MTSuccess)$
grd6 : $mt \notin ran(MTSuccess)$
grd7 : $rts = dep(idata)^{-1}[\{mt\}]$
grd8 : $mts \subseteq mtasks(idata)$
grd9 : $\exists rt1 \cdot rt1 \in rts \land (rt1 \mapsto mt \mapsto mw) \in ExecMT$

then

act1 : $MTSuccess := MTSuccess \cup (rts \times \{mt\})$
act2 : $MTProcess_result(mt) := mworker_result(mw)(mt)$

end

Event $RTSplit \mathrel{\widehat{=}}$

extends $RTSplit$

any

rt
mts
presult

where

grd2 : $\texttt{rt} \in \texttt{rtasks}(\texttt{idata})$
grd3 : $\texttt{rt} \notin \texttt{RTSplit}$
grd4 : $\texttt{mts} \subseteq \texttt{ran}(\texttt{MTSuccess})$
grd5 : $\texttt{mts} = \texttt{dep}(\texttt{idata})[\{\texttt{rt}\}]$
grd6 : $\texttt{presult} = (\bigcup \texttt{mt} \cdot \texttt{mt} \in \texttt{mts} | \{\texttt{mt}\} \times \texttt{MTProcess_result}(\texttt{mt}))$
grd7 : $\texttt{rt} \in \texttt{dom}(\texttt{MTSuccess})$

then

act1 : $\texttt{RSplit} := \texttt{RSplit} \cup \{\texttt{rt}\}$
act2 : $\texttt{RSplit_result}(\texttt{rt}) := \texttt{rsplit}(\texttt{rt})(\texttt{presult})$

end

Event $RTProcess \mathrel{\widehat{=}}$

extends $RTProcess$

any

rt

where

grd1 : rt $\notin$ RTProcess

grd2 : rt $\in$ RSplit

then

act1 : RTProcess := RTProcess $\cup$ {rt}

act2 : Reduce_result(rt) := reduce(RTSplit_result(rt))

end

Event *RSplitReduceCommit* $\widehat{=}$

extends *RSplitReduceCommit*

when

grd1 : RTProcess = rtasks(idata)

grd2 : RSplitReduceCommit = FALSE

then

act1 : RSplitReduceCommit := TRUE

act2 : rsplit_result := RSplit_result

act3 : reduce_result := ($\bigcup$ rt·rt $\in$ rtasks(idata)|Reduce_result(rt))

end

Event *OutputMapReduce* $\widehat{=}$

extends *OutputMapReduce*

when

grd1 : RSplitReduceCommit = TRUE

grd2 : OutputMapReduce = FALSE

then

act1 : output := Combine(reduce_result)

act2 : OutputMapReduce := TRUE

end

END

MACHINE MapReduce2_m8

REFINES MapReduce2_m7

SEES MapReduce2_cnt

VARIABLES

`MapSplit`
`OutputMapReduce`
`idata`
`msplit`
`output`
`RSplitReduceCommit`
`rsplit_result`
`reduce_result`
`RTSplit`
`RTSuccess`
`RTSplit_result`
`Reduce_result`
`MTProcess_result`
`ExecMT`
`MTSuccess`
`mworker_result`
`AssignMT`
`RTok`
`rworker_result`

INVARIANTS

inv1 : $RTok \subseteq dom(rsplit) \times RW$
inv2 : $dom(RTok) \subseteq RTSplit$
inv3 : $rworker_result \in RW \pfun (RTask \pfun \mathbb{P}_1(OData))$
inv4 : $dom(rworker_result) = ran(RTok)$
inv5 : $MapSuccess = \varnothing \Rightarrow dom(RTok) = \varnothing$
inv6 : $RTSuccess = RTProcess$
inv7 : $RTSuccess \subseteq dom(RTok)$
inv8 : $\forall rt, rw \cdot rt \mapsto rw \in RTok \Rightarrow rw \in dom(rworker_result) \land rt \in dom(rworker_result(rw))$
inv9 : $\forall rw, rt \cdot rt \mapsto rw \in RTok \Rightarrow rworker_result(rw)(rt) = reduce(RTSplit_result(rt))$

EVENTS

Initialisation

begin

act1 : $output :\in OData$
act2 : $idata :\in IData$
act3 : $MapSplit := FALSE$

act4 : $OutputMapReduce := FALSE$
act5 : $msplit := \varnothing$
act6 : $RSplitReduceCommit := FALSE$
act7 : $rsplit_result := \varnothing$
act8 : $reduce_result := \varnothing$
act10 : $RTSplit := \varnothing$
act11 : $RTSuccess := \varnothing$
act12 : $Reduce_result := \varnothing$
act13 : $RTSplit_result := \varnothing$
act14 : $MTSuccess := \varnothing$
act15 : $MTProcess_result := \varnothing$
act16 : $ExecMT := \varnothing$
act17 : $mworker_result := \varnothing$
act18 : $AssignMT := \varnothing$
act19 : $RTok := \varnothing$
act20 : $rworker_result := \varnothing$

end

Event *MapSplit* $\widehat{=}$
extends *MapSplit*

when

grd1 : $\texttt{MapSplit} = \texttt{FALSE}$

then

act1 : $\texttt{MapSplit} := \texttt{TRUE}$
act2 : $\texttt{msplit} := \texttt{MSplit}(\texttt{idata})$

end

Event *AssignMT* $\widehat{=}$
extends *AssignMT*

any

rt
mt
mw

where

grd1 : $\texttt{MapSplit} = \texttt{TRUE}$
grd2 : $\texttt{rt} \in \texttt{rtasks}(\texttt{idata})$
grd3 : $\texttt{mt} \in \texttt{mtasks}(\texttt{idata})$
grd4 : $\texttt{mw} \in \texttt{MW}$
grd5 : $\texttt{rt} \mapsto \texttt{mt} \mapsto \texttt{mw} \notin \texttt{AssignMT}$
grd6 : $\texttt{mw} \notin \texttt{ran}(\texttt{AssignMT})$

then

act1 : $\texttt{AssignMT} := \texttt{AssignMT} \cup \{\texttt{rt} \mapsto \texttt{mt} \mapsto \texttt{mw}\}$

end

Event *ExecMT* $\widehat{=}$
extends *ExecMT*

any

mt
rt
mw

where

grd2 : $mt \in mtasks(idata)$
grd3 : $rt \in rtasks(idata)$
grd5 : $mw \notin dom(mworker_result)$
grd6 : $mt \notin ran(dom(ExecMT))$
grd7 : $\exists rt1 \cdot rt1 \in rtasks(idata) \land rt1 \mapsto mt \mapsto mw \in AssignMT$

then

act1 : $ExecMT := ExecMT \cup \{rt \mapsto mt \mapsto mw\}$
act2 : $mworker_result := mworker_result \cup \{mw \mapsto \{mt \mapsto map(msplit(mt))\}\}$

end

Event *MTSuccess* $\widehat{=}$

extends *MTSuccess*

any

mts
mt
rts
rt
mw

where

grd1 : $MapSplit = TRUE$
grd2 : $rt \in rtasks(idata)$
grd3 : $mts = dep(idata)[\{rt\}]$
grd4 : $mt \in mts$
grd5 : $rt \notin dom(MTSuccess)$
grd6 : $mt \notin ran(MTSuccess)$
grd7 : $rts = dep(idata)^{-1}[\{mt\}]$
grd8 : $mts \subseteq mtasks(idata)$
grd9 : $\exists rt1 \cdot rt1 \in rts \land (rt1 \mapsto mt \mapsto mw) \in ExecMT$

then

act1 : $MTSuccess := MTSuccess \cup (rts \times \{mt\})$
act2 : $MTProcess_result(mt) := mworker_result(mw)(mt)$

end

Event *RTSplit* $\widehat{=}$

extends *RTSplit*

any

rt
mts
presult

where

grd2 : $rt \in rtasks(idata)$
grd3 : $rt \notin RTSplit$

grd4 : $\texttt{mts} \subseteq \texttt{ran}(\texttt{MTSuccess})$
grd5 : $\texttt{mts} = \texttt{dep}(\texttt{idata})[\{\texttt{rt}\}]$
grd6 : $\texttt{presult} = (\bigcup \texttt{mt} \cdot \texttt{mt} \in \texttt{mts} | \{\texttt{mt}\} \times \texttt{MTProcess_result}(\texttt{mt}))$
grd7 : $\texttt{rt} \in \texttt{dom}(\texttt{MTSuccess})$

then

act1 : $\texttt{RTSplit} := \texttt{RTSplit} \cup \{\texttt{rt}\}$
act2 : $\texttt{RTSplit_result}(\texttt{rt}) := \texttt{rsplit}(\texttt{rt})(\texttt{presult})$

end

Event *RTok* $\widehat{=}$

any

rt
rw

where

grd1 : $rt \in RTSplit$
grd2 : $rw \in RW$
grd3 : $rt \mapsto rw \notin RTok$
grd4 : $rw \notin ran(RTok)$

then

act1 : $RTok := RTok \cup \{rt \mapsto rw\}$
act2 : $rworker_result := rworker_result \cup \{rw \mapsto \{rt \mapsto reduce(RTSplit_result(rt))\}\}$

end

Event *RTSuccess* $\widehat{=}$

refines *RTProcess*

any

rt
rw

where

grd1 : $rt \in RTSplit$
grd2 : $rt \notin RTSuccess$
grd3 : $rt \mapsto rw \in RTok$

then

act1 : $RTSuccess := RTSuccess \cup \{rt\}$
act2 : $Reduce_result(rt) := rworker_result(rw)(rt)$

end

Event *RSplitReduceCommit* $\widehat{=}$

refines *RSplitReduceCommit*

when

grd1 : $RTSuccess = rtasks(idata)$
grd2 : $RSplitReduceCommit = FALSE$

then

act1 : $RSplitReduceCommit := TRUE$
act2 : $rsplit_result := RTSplit_result$
act3 : $reduce_result := (\bigcup rt \cdot rt \in rtasks(idata) | Reduce_result(rt))$

end

Event *OutputMapReduce* $\widehat{=}$
extends *OutputMapReduce*
when
grd1 : RSplitReduceCommit = TRUE
grd2 : OutputMapReduce = FALSE
then
act1 : output := Combine(reduce_result)
act2 : OutputMapReduce := TRUE
end
END

MACHINE MapReduce2_m9

REFINES MapReduce2_m8

SEES MapReduce2_cnt

VARIABLES

`MapSplit`
`OutputMapReduce`
`idata`
`msplit`
`output`
`RSplitReduceCommit`
`rsplit_result`
`reduce_result`
`RTSplit`
`RTSuccess`
`RTSplit_result`
`Reduce_result`
`MTProcess_result`
`ExecMT`
`MTSuccess`
`mworker_result`
`AssignMT`
`ExecRT`
`rworker_result`
`AssignRT`

INVARIANTS

`inv1` : $AssignRT \subseteq dom(rsplit) \times RW$
`inv2` : $ExecRT = RTok$
`inv3` : $dom(AssignRT) \subseteq RTSplit$
`inv4` : $ExecRT \subseteq AssignRT$

EVENTS

Initialisation

begin

`act1` : $output :\in OData$
`act2` : $idata :\in IData$
`act3` : $MapSplit := FALSE$
`act4` : $OutputMapReduce := FALSE$
`act5` : $msplit := \varnothing$
`act6` : $RSplitReduceCommit := FALSE$
`act7` : $rsplit_result := \varnothing$
`act8` : $reduce_result := \varnothing$

act10 : $RTSPlit := \varnothing$
act11 : $RTSuccess := \varnothing$
act12 : $Reduce_result := \varnothing$
act13 : $RTSplit_result := \varnothing$
act14 : $MTSuccess := \varnothing$
act15 : $MTProcess_result := \varnothing$
act16 : $MTok := \varnothing$
act17 : $mworker_result := \varnothing$
act18 : $AssignMT := \varnothing$
act19 : $ExecRT := \varnothing$
act20 : $rworker_result := \varnothing$
act21 : $AssignRT := \varnothing$

end

Event *MapSplit* $\widehat{=}$
extends *MapSplit*

when

grd1 : MapSplit = FALSE

then

act1 : MapSplit := TRUE
act2 : msplit := MSplit(idata)

end

Event *AssignMT* $\widehat{=}$
extends *AssignMT*

any

rt
mt
mw

where

grd1 : MapSplit = TRUE
grd2 : rt $\in$ rtasks(idata)
grd3 : mt $\in$ mtasks(idata)
grd4 : mw $\in$ MW
grd5 : rt $\mapsto$ mt $\mapsto$ mw $\notin$ AssignMT
grd6 : mw $\notin$ ran(AssignMT)

then

act1 : AssignMT := AssignMT $\cup$ {rt $\mapsto$ mt $\mapsto$ mw}

end

Event *ExecMT* $\widehat{=}$
extends *ExecMT*

any

mt
rt
mw

where

grd2 : mt ∈ mtasks(idata)
grd3 : rt ∈ rtasks(idata)
grd5 : mw ∉ dom(mworker_result)
grd6 : mt ∉ ran(dom(ExecMT))
grd7 : ∃rt1·rt1 ∈ rtasks(idata) ∧ rt1 ↦ mt ↦ mw ∈ AssignMT

then

act1 : ExecMT := ExecMT ∪ {rt ↦ mt ↦ mw}
act2 : mworker_result := mworker_result∪{mw ↦ {mt ↦ map(msplit(mt))}}

end

Event *MTSuccess* ≙

extends *MTSuccess*

any

mts
mt
rts
rt
mw

where

grd1 : MapSplit = TRUE
grd2 : rt ∈ rtasks(idata)
grd3 : mts = dep(idata)[{rt}]
grd4 : mt ∈ mts
grd5 : rt ∉ dom(MTSuccess)
grd6 : mt ∉ ran(MTSuccess)
grd7 : rts = dep(idata)$^{-1}$[{mt}]
grd8 : mts ⊆ mtasks(idata)
grd9 : ∃rt1·rt1 ∈ rts ∧ (rt1 ↦ mt ↦ mw) ∈ MTok

then

act1 : MTSuccess := MTSuccess ∪ (rts × {mt})
act2 : MTProcess_result(mt) := mworker_result(mw)(mt)

end

Event *RTSplit* ≙

extends *RTSplit*

any

rt
mts
presult

where

grd2 : rt ∈ rtasks(idata)
grd3 : rt ∉ RTSplit
grd4 : mts ⊆ ran(MTSuccess)
grd5 : mts = dep(idata)[{rt}]
grd6 : presult = (⋃mt·mt ∈ mts|{mt} × MTProcess_result(mt))
grd7 : rt ∈ dom(MTSuccess)

then

act1 : $\texttt{RTSplit} := \texttt{RTSplit} \cup \{\texttt{rt}\}$

act2 : $\texttt{RTSplit_result}(\texttt{rt}) := \texttt{rsplit}(\texttt{rt})(\texttt{presult})$

end

Event *AssignRT* $\widehat{=}$

any

rt

rw

where

grd1 : $rt \in RTSplit$

grd2 : $rw \in RW$

grd3 : $rt \mapsto rw \notin AssignRT$

grd4 : $rw \notin ran(AssignRT)$

then

act1 : $AssignRT := AssignRT \cup \{rt \mapsto rw\}$

end

Event *ExecRT* $\widehat{=}$

refines *RTok*

any

rt

rw

where

grd1 : $rt \in RTSplit$

grd2 : $rw \in RW$

grd3 : $rt \mapsto rw \notin ExecRT$

grd4 : $rw \notin ran(ExecRT)$

grd5 : $rt \mapsto rw \in AssignRT$

then

act1 : $ExecRT := ExecRT \cup \{rt \mapsto rw\}$

act2 : $rworker_result := rworker_result \cup \{rw \mapsto \{rt \mapsto reduce(RTSplit_result(rt))\}\}$

end

Event *RTSuccess* $\widehat{=}$

extends *RTSuccess*

any

$\texttt{rt}$

$\texttt{rw}$

where

grd1 : $\texttt{rt} \in \texttt{RTSplit}$

grd2 : $\texttt{rt} \notin \texttt{RTSuccess}$

grd3 : $\texttt{rt} \mapsto \texttt{rw} \in \texttt{ExecRT}$

then

act1 : $\texttt{RTSuccess} := \texttt{RTSuccess} \cup \{\texttt{rt}\}$

act2 : $\texttt{Reduce_result}(\texttt{rt}) := \texttt{rworker_result}(\texttt{rw})(\texttt{rt})$

end

Event *RSplitReduceCommit* $\widehat{=}$
extends *RSplitReduceCommit*
when
grd1 : $\texttt{RTSuccess} = \texttt{rtasks}(\texttt{idata})$
grd2 : $\texttt{RSplitReduceCommit} = \texttt{FALSE}$
then
act1 : $\texttt{RSplitReduceCommit} := \texttt{TRUE}$
act2 : $\texttt{rsplit_result} := \texttt{RSplit_result}$
act3 : $\texttt{reduce_result} := (\bigcup \texttt{rt} \cdot \texttt{rt} \in \texttt{rtasks}(\texttt{idata}) | \texttt{Reduce_result}(\texttt{rt}))$
end
Event *OutputMapReduce* $\widehat{=}$
extends *OutputMapReduce*
when
grd1 : $\texttt{RSplitReduceCommit} = \texttt{TRUE}$
grd2 : $\texttt{OutputMapReduce} = \texttt{FALSE}$
then
act1 : $\texttt{output} := \texttt{Combine}(\texttt{reduce_result})$
act2 : $\texttt{OutputMapReduce} := \texttt{TRUE}$
end
END

Turku
Centre *for*
Computer
Science

Joukahaisenkatu 3-5 B, 20520 Turku, Finland | www.tucs.fi

University of Turku
- Department of Information Technology
- Department of Mathematics

Åbo Akademi University
- Department of Information Technologies

Turku School of Economics
- Institute of Information Systems Sciences

ISBN 978-952-12-3006-6
ISSN 1239-1891