SDP Memo: SDP Data Driven Architecture I: Justification

Document number	SKA-TEL-SDP-0000066
Document type	REP
Revision	01C
Author	Bojan Nikolic, Montse Farreras
Release date	2016-04-08
Document classification	Unrestricted
Status	Draft

Lead author:

Name	Designation	Affiliation
Bojan Nikolic	SDP Project Engineer	University of Cambridge
Signature & Date:		

Released by:

Name	Designation	Affiliation
Paul Alexander	SDP Project Lead	University of Cambridge
Signature & Date:		

Version	**Date of issue**	**Prepared by**	**Comments**
0.1	2016-04-01	Bojan Nikolic	Initial circulated draft
0.2	2016-04-06	Bojan Nikolic	Correct references and spelling
0.3	2016-04-07	Montse Farreras	Incorporate further clarifications
01C	2016-04-08	Bojan Nikolic	Released for $\triangle$PDR

ORGANISATION DETAILS

Name	Science Data Processor Consortium

Table of Contents

List of abbreviations

ASKAP	Australian Square Kilometre Array Pathfinder
CASA	Common Astronomy Software Applications
LTM	Local Telescope Model
LTS	Local Telescope State
O/S	Operating System
PDR	Preliminary Design Review
PGAS	Partitioned Global Address Space
SDP	Science Data Processor
SKA	Square Kilometre Array

Applicable and reference documents

Applicable Documents

The following documents are applicable to the extent stated herein. In the event of conflict between the contents of the applicable documents and this document, *the applicable documents* shall take precedence.

Reference Number	Reference
AD01	SKA-TEL-SDP-0000013 – SDP Element Architecture Design, 2016

Reference Documents

The following documents are referenced in this document. In the event of conflict between the contents of the referenced documents and this document, *this document* shall take precedence.

Reference Number	Reference
RD01	J. Dean and Ghemawat, S.: MapReduce: simplified data processing on large clusters, Communications of the ACM, vol. 51, nr. 1, pp. 107–113, 2008
RD02	M. J. Flynn: Some Computer Organizations and Their Effectiveness, IEEE Transactions on Computers, vol. C-21, no. 9, pp. 948–960, doi:10.1109/TC.1972.5009071, 1972
RD03	SKA-TEL-SDP-0000013 – SDP Architecture Design, rev. 1.1, 2015
RD04	SKA-TEL-SDP-0000026 – SDP Local Monitoring and Control Design, 2016
RD05	SKA-TEL-SDP-0000072 – Exploring the use of an existing big-data processing system architecture for the SDP, 2015
RD06	SKA-TEL-SDP-0000085 – Horizontal Prototyping Interim Report, 2016
RD07	SKA-TEL-SDP-0000066 – The potential of a data flow architecture for the SDP, 2015
RD08	SKA-TEL-SDP-0000015 – SDP Execution Framework Design, 2016

1 Introduction

In this memo we summarise the reasons why a data-driven architecture has been adopted for the processing functionality of the SKA SDP element. The reader is also referred to RD07, AD01 and RD08 for details about the architecture and further justifications of the choice.

2 Summary of key points of the data driven architecture

AD01 and RD08 present a detailed design for the data-driven architecture. In this memo we seek to justify the key aspects, which can be summarised as follows:

1. The SDP processing is divided into a set of distinct tasks[1]. Each task:
   (a) Has an explicitly declared complete list of inputs and outputs
   (b) The execution of any task is only possible after all inputs are complete and the inputs never change during execution
   (c) The tasks' outputs are never used by other tasks during their execution, only after they have terminated
   (d) There is no mechanisms for tasks to share or exchange any data directly other than through their declared inputs and outputs[2]
   (e) There is no recursive task structure, i.e., a task does not itself consist of other tasks; instead a task is opaque at architectural level
2. The sequencing and assignment of tasks to computational resources is *not* done during the implementation of the processing software. Instead only the *dependencies* between tasks are defined during implementation; specifically these dependencies are defined by specifying *data dependencies*, i.e., the outputs of certain tasks are used as inputs of other tasks.
3. The final sequencing and assignment of tasks to computational resources is done only during *run-time* by the computer and the execution environment itself.

The SDP architecture departs from this simplified summary in some significant ways which are discussed in Section 6 after the initial justification of the overall approach.

3 Comparison to alternatives

3.1 Explicit sequencing, assignment and synchronisation

An obvious alternative is the programming used for almost all large simulation applications, as well as in the domain of radio astronomy, the ASKAPSoft package. This model can be summarised as follows:

1. A clear task structure is not enforced by the architecture
2. The division of computation between the computational resources is explicit. It is specified during the implementation of the program, and a segregation of this division of computation from the details of computation is not enforced

[1]The intended use of *task* is in the sense of RD02
[2]But see Section 6 for an important departure of the SDP architecture from this principle

3. The synchronisation of computation on different computational resources is explicit, either pair-wise or over all of the resources, i.e., bulk-synchronous. There is no enforced segregation of this synchronisation specification from the domain specific parts of the code.

This class of architectures includes most traditional MPI-based applications as well as most PGAS and similar applications. We will below refer to this architecture as the *explicit control flow architecture*.

3.2 MapReduce

MapReduce is the most high-profile example of the recent and currently emerging architectures for processing large data-sets in the domains of commerce, internet technology and similar fields (RD01). We believe the most relevant aspect of the MapReduce model can be summarised as follows:

1. Data is divided into a set of keys each of which has one or more associate values.
2. Processing is divided into 'mapping' and 'reduction' tasks.
3. Tasks have access only to their specific input and output data and no other communication or sharing of state between tasks is possible. The input data are:

   For mapper tasks Pairs of key and individual values

   For reduction tasks Key and set of all associated value pairs
4. There is one mapping and one reduction stage that operate in separate spaces of keys
5. The assignment of tasks to computational resources, sequencing and synchronisation are handled by the run-time system using the implicit data dependencies as defined by the keys

Other discussion of similar architectures is presented in RD05. We did not have the time to consider other architectures emerging from these fields in the preparation of this document but this is something that the SDP needs to do in the future and keep monitoring throughout the design process.

4 Justification of the SDP choice

The immediate objectives in the choice of architecture for the processing software are given by AD01 and can be summarised as follows:

1. The architecture needs to be capable of implementing all of the processing functionality that the SDP needs to implement, defined as at least all of the current best-practice approaches to radio-interferometric data reduction
2. Achieve scaling of SDP software without strong coupling to the SDP hardware architecture or system size
3. Achieve reasonable efficiency, in particular with respect to load balancing
4. Achieve architectural separation of domain specific functionality from parts of the system that enable scalability and flexibility

We first describe why the alternatives presented in Section 3 do not satisfy these objectives, then examine how the chosen architecture does enable these objectives. Finally we present a summary of drawbacks of the chosen architecture.

4.1 Evaluation of alternatives against the objectives

First we review the explicit control flow architecture summarised in Section 3.1 against the objectives. This architecture can support all the processing functionality required by the SDP as it is a fully general system that is a superset of functionality used today in standard astronomy applications such as CASA.

This architecture also provides in principle a route to efficiency (objective 3), since a carefully chosen sequencing, assignment and synchronisation, together with low-jitter O/S and hardware should produce reasonable efficiency for one specific problem. However, in practice getting this efficiency may be very difficult because:

- High efficiency requires optimisation by hand of sequencing, assignment and synchronisation which may be interspersed in the domain-specific code. This is slow work, requires a high degree of skill, and is error prone.
- Differences between observation types or data reduction strategies are likely significant so that different optimisations would need to be produced, leading to a divergent code base
- If there is significant jitter, i.e., unpredictability in the duration of execution of certain tasks, due to either O/S or hardware means that any ahead-of-time selected scheduling of the computation will lead to diminished efficiency.

Finally, we believe that this architecture does not provide a route to achieving objectives 2 and 4 because:

- The assignment of computation to computational resources and their synchronisation is explicitly specified in the program implementation. This creates a strong coupling to the hardware; for example an assignment which is efficient for a particular high-performance interconnect may not be suitable for a more cost-effective interconnect.
- There is no architectural segregation in this model between domain-specific functionality and the software that enables scalability and flexibility.

The second architecture that we review in this section is the Map-Reduce architecture. This architecture as presented in Section 3.2 can not meet the first objective, i.e., it can not implement the processing required for the SDP. For example iteration (which is a key part of calibration and de-convolution in interferometric imaging) can not be represented in this architecture.

A more fundamental issue with using the MapReduce architecture is that SDP input, intermediate and output data are regularly structured, which does not map well to the principle of key-based handling of data. While regular structures can be encoded in keys, these will often create a strong coupling to software implementations and, we believe, lead to difficult optimisation for different hardware architectures.

4.2 Evaluation of the data-driven architecture

In this section we discuss the data-driven architecture, evaluated against the objectives given in the beginning of this section.

4.2.1 Implementability of the SDP algorithms

The summary given in Section 2 is a fairly general architecture that can represent most computations. The most difficult computations to represent in this architectures are computations to respond to external commands or adjustments, or computations that are heavily data dependent, e.g., where the program is updated during computation. Since neither of these appear relevant to the SKA SDP, we believe that the architecture as described in Section 2 is capable of implementing all of the computations needed for the SDP. However, this issue does need continuous review, especially as the design of the data-driven framework is further progressed.

A simple SDP-specific example of a difficulty for data driven architectures is stopping a de-convolution loop based on a convergence criterion: here it is impossible to know which tasks can be executed after the task containing the convergence test until this task is in fact complete and the values of its results are known. Sub-classes of data-driven architecture that do not have *data-dependent control flow* capability, i.e., those architectures that do not allow the selection of future tasks to be run based on the output *values* of past tasks, can not implement such convergence criterion stopping.

Recent prototyping in the SDP has aimed to address the implementability of SDP computations on data-driven architectures, however we have not had the time to review the results in detail for this memo. A critical review will be presented in a forthcoming memo.

4.2.2 Scaling and avoiding strong coupling to hardware

A number of reasons have been proposed why the data-driven approach will benefit the scaling of SDP software and reduce the coupling to the hardware architecture. The ones we are aware of are:

1. The tasks are placed on computational resources automatically. This reduces the coupling to hardware architecture and system sizing; for example the application may be run on two systems with different (by a significant factor) number of nodes without having to adjust of the application code.

2. A clever implementation of the data-driven approach can determine the sequencing of tasks on each node without necessarily interacting with a single master node. This reduces the scaling bottleneck associated with such a single master node.

3. The placement of tasks can be automatically optimised to reduce the network data transfer requirements, both to access the input data and also to pass intermediate data products. This reduces the coupling to hardware architecture and also reduces the scaling limitations due to the limits of networking hardware.

We are not aware that any of these reasons have yet been developed past a basic theoretical argument for substantially complete SDP or SDP-like workloads. Some demonstrations of scalability have however been done in SDP prototypes of *simplified* radio-interferometric processing which will be summarised in a forthcoming memo. A further development of the above reasons and further demonstrations should form an important part of the SDP plans for future work.

There is developing published evidence from other applications for scalability of data-driven approaches similar to that summarised in Section 2. Examples such existing systems are Parsec and SWIFT/T. Some references and relevant information on these can be found in

RD03. A proper review of these published results with commentary of relevant aspects for the SDP would be a worthwhile part of the near-term SDP work plan.

Through pen and paper analysis we have identified some parts of the SDP computations that have complex data dependencies and therefore may pose problems for a data-driven approach. The most significant of these appears to be the transitions between the calibration and imaging stages of processing. This is currently very briefly described in a short internal SDP note. These identified SDP computations that may not scale well in the data-driven approach will be a subject of a forthcoming memo where they will be described more fully. Each of these will need to be addressed in the future SDP work plan.

4.2.3 Efficiency

We have identified the following reasons why the data-driven may help achieve reasonable efficiency:

1. The initial static placement of tasks can be optimised automatically by the computer, taking into account information such as available system size, network architecture, record of their performance in the past
2. The placement of tasks can further be adjusted dynamically during the computation, to take into account the actual performance for the problem at hand
3. Data locality can be optimised leading to higher efficiency when bandwidth or latency of network transfers are limiting factors
4. Data dependencies between tasks are computed so that a task can run as soon as his input data is ready. This avoids unnecessary synchronisation points and can help speed up the critical path of the application leading to an overall performance improvement.

As is the case for scaling (Section 4.2.2) there are no demonstrations of the above points for substantially complete SDP workloads but some prototyping of simplified processing has been done. Again, this will be reviewed in a future memo.

4.2.4 Resilience and fault tolerance

We have identified the following reasons why the data-driven approach may help achieve resilience and fault tolerance:

1. The tasks are dynamically scheduled by the software on the available resources. This allows the system to be adaptive to changes on the architecture.
2. The completely specified data dependencies in this architecture open the possibility of non-checkpoint based failure recovery, i.e., re-starting of failed tasks or handling tasks through the non-precious data concept discussed in RD03. These alternative failure recovery mechanism may have better scaling properties than check-pointing.

Again there is no demonstration of the above points in the context of the SDP and it is an important part of our future work.

4.2.5 Separation of domain specific functionality

The data-driven approach architecturally separates its functionality into three domains:

- The specification of the processing within a task
- The specification of the data dependencies between tasks
- The algorithms which interpret data dependencies, place tasks onto computational resources and sequence the tasks

The first two are clearly domain specific since they depend on the details of the processing that needs to be done to obtain output data products of good quality. Some of the prototyping described in RD06 has relevant findings on the division of functionality between the first two items.

The third item in the above list *can* however be free of domain knowledge. What however still needs to be demonstrated is that these algorithms can be constructed without domain-specific knowledge and still give reasonably good scalability and efficiency for the data-driven system. For example, domain-specific knowledge suggests that it is beneficial to place tasks processing neighbouring frequencies on the same computational resource. A task placement algorithm without domain knowledge would however need to analyse the data dependence graph to come to the same conclusion.

5 Drawbacks of the data-driven architecture

We list below some suspected technical drawbacks of the data-driven architecture.

1. Difficulty in steering the processing of data: it is difficult to incorporate into this architecture functionality to heuristically change the data processing strategy during the processing or to allow real-time steering of the computation by an operator. This is *not* a current SDP requirement, but if it becomes necessary in the future it may be difficult to adopt the data-driven architecture to achieve this.
2. Regarding memory use. Because of the large amounts of data to be processed, memory bandwidth is likely to be a limiting factor for the SDP system. On top of this, the data-driven architecture does not provide a good mechanism to use a shared memory space, being limited to the task level. Because of that reason, if parallelism at a large scale needs to be exploited, copies of data may be required leading to an increase in the memory use for task input and output data, which may be quite significant.

These will be investigated in more detail in the future and if needed will form part of the post-PDR SDP work.

6 Departures from the summary description

Here we highlight one departure of the SDP architecture from the simplified summary description in Section 2: the Local Telescope State (LTS) and Local Sky Model (LTM) functions (see their detailed description in AD01 and RD04). These functions provide data which are essential in most stages of the SDP processing, and additionally these functions accept and accumulate intermediate and final results of some of the SDP computations. These two functions are architected as request/reply *services*, a model entirely different from the model presented in Section 2.

This architecture opens the following areas which will need to be evaluated in the future:

1. The scalability and performance of the LTS and LSM services
2. The complexity of implementing both service and data-driven architectures side-by-side in the same application
3. The interaction of the service-based functions with the data-driven approach. For example without careful epoch-tagging of data in the services, the results of the data-driven processing will not be deterministic[3]. As another example, will this architecture significantly decrease the performance of the data-driven architecture?

7 Final remarks

The data-driven SDP architecture offers an opportunity for a stable foundation for the SKA science processing software for a long time into the future. It clearly carries significant risks including (on time) implementation risks, difficulty in maintenance, failing to achieve performance goals and others. Some discussion of the risks is presented by RD08.

The SDP consortium additionally intends to focus on building evidence that, if implemented, the approach will actually meet the objectives. Building this evidence base will require developing the objectives into quantitative requirements for the appropriate SDP sub-systems, followed by a systematic validation that these are likely to be met by the final system.

[3]Thanks to Peter Wortmann for pointing this out.