SDP Memo 038: Pipeline Working Sets and Communication

Document Number ..... SDP Memo 038
Document Type ..... MEMO
Revision 2
Author ..... Peter Wortmann
Release Date 2017-12-14
Document Classification ..... Unrestricted
Status ..... Draft

Lead Author	Designation	Affiliation
Peter Wortmann	Research Associate	University of Cambridge
Signature & Date:		

Revision	Date of issue	Prepared by	Comments
1	2017-12-01	Peter Wortmann	Initial publication
2	2017-12-14	Peter Wortmann	Revised calibration after discussion with Stefano Salvini and Tim Cornwell

SDP Memo Disclaimer

The SDP memos are designed to allow the quick recording of investigations and research done by members of the SDP. They are also designed to raise questions about parts of the SDP design or SDP process. The contents of a memo may be the opinion of the author, not the whole of the SDP.

Table of Contents

1 Introduction

The pipelines of the SKA SDP are clearly very data intensive: Even for a large cluster the input rate of 0.4 TB/s is not trivial to absorb and process. However, the real problem emerges in combination with the nature of imaging and calibration. Serving such astronomical algorithms involves extracting as much information as possible from the input data concerning certain time slot, frequency channel, or sky directions. Unfortunately, this generally boils down to exceptionally large queries into the already-large data sets. In fact, imaging is the worst case here, as basically every single input visibility contains a certain amount of information about any given sky direction.

This means that we have to be extremely careful with how exactly we move the data through the system. Algorithms that appear trivial on paper can easily turn out to be basically impossible to implement because of the data reorganisation involved. Worse, effective self-calibration pipelines involve running many different types of demanding imaging and calibration algorithms without much common ground in terms of access or aggregation patterns. So unless we prove otherwise, it would seem likely that we might run into subtle incompatabilities down the road.

For this memo, we will therefore focus on these data reorganisations and aggregations and attempt to show that we can move data around in a way that

- we never run out of working memory,
- we never exceed communication limits, and
- we actually finish within in the designated time.

We will see that these goals are in strong competition with each other. However, characterising these trade-offs exactly yields us a lot of useful restrictions on both algorithm choice as well as the mapping to the compute hardware. By the end of this memo, we will have restricted the design space enough to arrive at some pretty concrete recommendations.

1.1 Motivation

As a starting point, consider that 0.4 TB/s of input visibility data adds up to about 1.4 PB of data per hour. So if we assume to have roughly 1400 compute nodes available, this means that even

holding just this short observation in memory would require around 1 TB of working memory. And it gets worse: For calibration and deconvolution we would need to predict model visibilities to compare with the measurement data, which might multiply the amount of visibility data up quite a bit further. And this is even before we starting considering images and calibration data.

So unless we get strategic with how we handle data, this could easily multiply up the working to multiple TB worth of data, with no obvious upper bound in sight. This actually seems to be a very real trend with current telescopes: The processing model of the SKA precursor MeerKAT places key functions on four nodes with RAM in the Terabyte scale. Similar things seem to be happening for the Askap precursor as well.

For the Science Data Processor this is clearly a problem, as we need to do this on a much larger scale. Even for 2022 hardware Graser (2016) does not budget for more than 500 GB of memory per node. The ability to run SDP pipelines on this sort of hardware is clearly not going to happen by accident, we need guarantees.

1.2 Assumptions

Before we start with the main body of the memo, let us get a few assumptions out of the way. The numbers for this analysis are based chiefly on parametric model (Bolton et al., 2016) predictions for data sizes and compute complexity.

We especially adopt the parametric model nomenclature of grouping pipelines producing images as "imaging pipelines", even if they consist of prediction as well as invert steps. This is convenient as predict and invert is mostly symmetric if we consider FFT-based methods to be required – which we should at the targeted scale of the SKA. In the following, when we talk about imaging we are therefore referring to the FFT predict/invert pair with all supporting infrastructure such as reprojections, facet splits, phase rotation, gridding and degridding.

However, we make a number of assumptions on top of the parametric model:

- We assume that for continuum imaging we use three Taylor terms, so $N_{\text{tt}} = 3$. After talking with several astronomers, this seems to be a more realistic choice than the original value of $N_{\text{tt}} = 5$, see for example discussion in Nikolic et al. (2016). We will see that this number scales basically everything, so we have to choose it carefully.

- Normally the parametric model implicitly allows a pipeline to run for as long as the length of the observation, reflecting a "steady state" model of operation. However, in reality we will want to run more than just a single pipeline on the observation data, therefore we introduce a top-level factor Q_{speed} that scales the time we have to finish processing relative to the length of the observation.

- We will fix the snapshot length at $t_{\text{snap}} = 600\,\text{s}$. Background is that we expect to be handling w-terms using a mixture of w-stacking as used by Offringa et al. (2014) and w-snapshots (Cornwell et al., 2012). This should make it much easier to choose snapshot size for architectural reasons, and a value around ten minutes seems to work well.

Furthermore, throughout this memo we will only consider the Mid1 band for the SKA1 Mid telescope. This is because all Mid bands generally behave very similarly for the purpose of the SDP, with the Mid1 band the narrow cost leader. See Table 4 at the end of the document for full numbers.

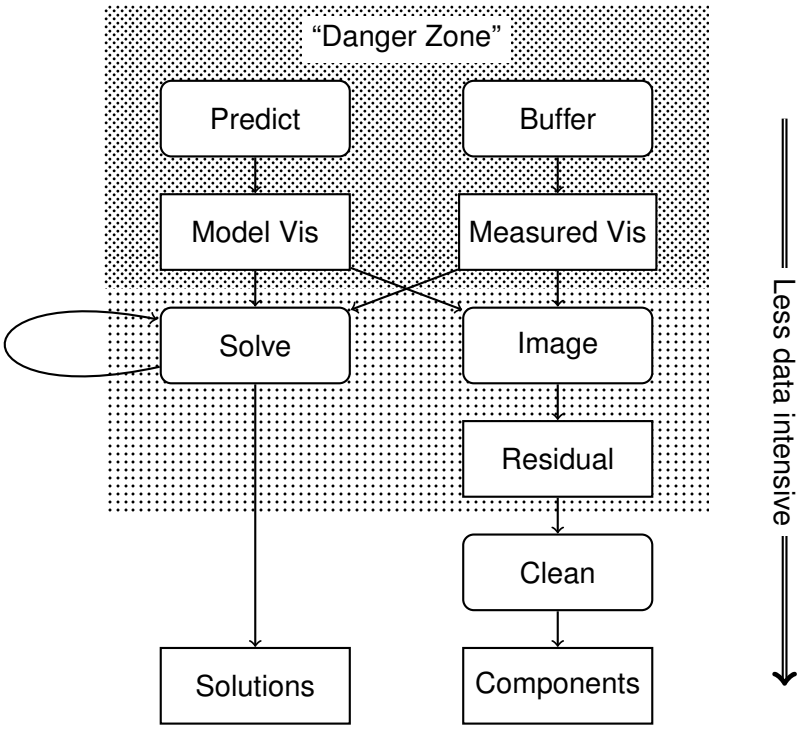

Figure 1: Continuum Pipeline Major Loop Structure

2 Visibilities

Sadly, the working set size we alluded to in subsection 1.1 is not a fantasy: As Figure 1 shows, a major loop iteration is a linear data reduction process with a very "top-heavy" structure. There are no natural synchronisation points that would suggest that we are "done" with a given type of data entirely. So a naïve execution of the pipeline might use too much of the available parallelism (a well-known effect in programs with a data flow representation, see Culler and Arvind, 1988), and therefore stay in the dangerous zone for too long.

Note that in Figure 1 we have drawn this zone to end mostly with visibilities. The reasoning is that both images and calibration solutions are less problematic, as they can be aggregated rather aggressively. We will explore that in detail in section 3 and section 4 respectively. An important exception here is extreme spectral pipelines (like DPrepC), for which we cannot aggregate images as much. We will have a closer look at that case in section 5.

2.1 Visibility Rates

So let us return to the visibility working set argument from the Motivation section. What are our tolerances here exactly? We know that we will have to loop over all data at minimum N_{major} times for the major loop, and we will likely have to do it faster than we received the data originally. So we know that in Figure 1 the measured visibility data will be read from the buffer at the following rate per node:

$$R_{\text{vis/node}} = \frac{R_{\text{vis}} Q_{\text{speed}} N_{\text{major}}}{N_{\text{node}}} \approx 4.414\,\text{GB/s} \quad \text{for SKA1 Low}$$
$$\approx 4.805\,\text{GB/s} \quad \text{for SKA1 Mid1}$$

If we assume that we need to produce N_{predict} separate model visibilities at the same time – such as for major loop subtraction or calibration – model visibilities will contribute an even more significant influx of data. The amount of prediction needed clearly depends on the experiment, but even with just $N_{\text{predict}} = 4$ this would add up to:

$$\begin{aligned} R_{\text{predict/node}} = N_{\text{predict}} R_{\text{vis/node}} &\approx 17.65\,\text{GB/s} \quad \text{for SKA1 Low} \\ &\approx 19.22\,\text{GB/s} \quad \text{for SKA1 Mid1} \end{aligned}$$

2.2 Time to Work

From this we can now easily calculate how much memory we need to be able to "work" on a given piece of data for a certain time:

$$M_{\text{vis/node}} = t_{\text{work}}(R_{\text{predict/node}} + R_{\text{vis/node}})$$

$$t_{\text{work}} = \frac{N_{\text{node}}}{R_{\text{predict/node}} + R_{\text{vis/node}}}$$

Let us assume that we can allocate 64 GB of node memory for storing visibilities, this yields us 2.83 s (Low) or 2.92 s (Mid1) respectively for leaving the "danger zone" before memory runs out. This is a very small amount of time, given that:

- This must include time needed for either generating the data, or loading it from either storage or over network
- There must be some slack due to imperfect scheduling or congestion
- *All* computation on that data must be finished before the time runs out

We can extend this time window by increasing node memory. We can actually easily calculate the cost of that: At a price of memory of 0.5 €/GB extending this time window by a second would add roughly $15\,000$ €/s to the budget (and 4.5 kW assuming 0.3 W/GB) (Graser, 2016).

2.3 Streaming

This heavily suggests a "streaming" kind of processing model for visibilities: Explicitly work in a sequential fashion, aligned with the order that data gets loaded into memory. So ideally, we would:

1. Stream visibilities in from the buffer
2. Predict alongside, synchronised using e.g. back-pressure (see subsection 3.9)
3. Do low-latency local processing
4. Output "processed" visibility data into a number of output buffers
5. Discard raw visibilities

Keeping these processes in alignment will require some careful intermediate caching as shown in Figure 2. This is especially important because as we will see, prediction, imaging and calibration can *not* be distributed in a way that aligns with visibility streaming.

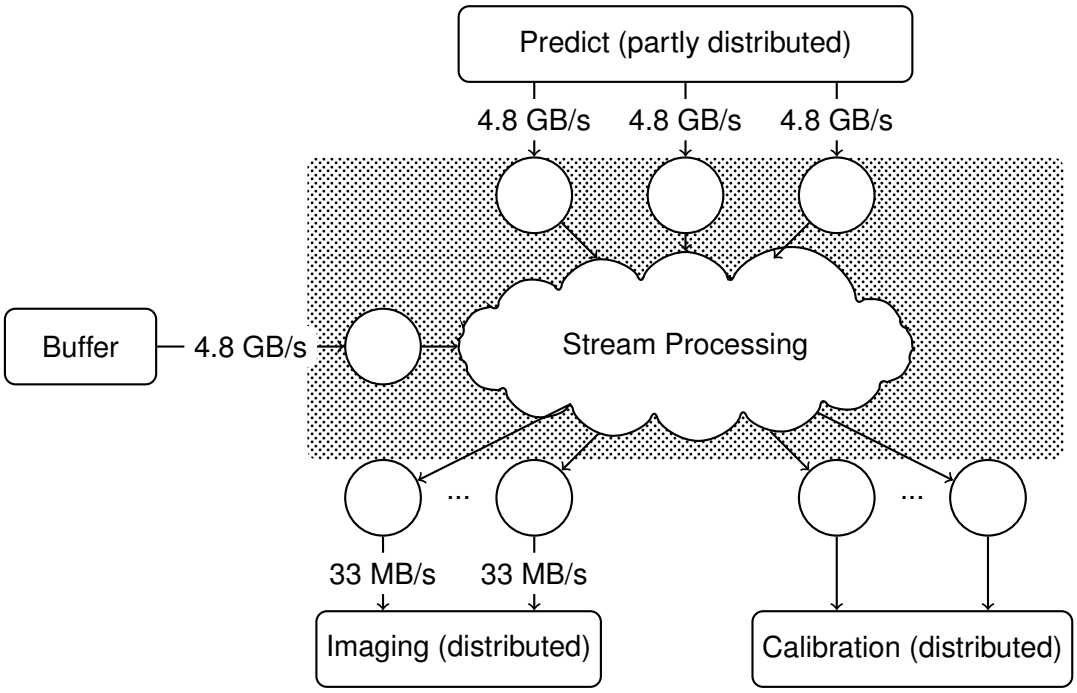

Figure 2: Visibility Streams through a Node

2.4 Stream Processing

By doing processing in this way, we radically reduce the window we have into the data at any given time. Fortunately, there is quite a bit of processing that we could do in this fashion:

- *Subtraction:* Clearly subtracting, say, the model from the measurements is cheap and context-free. This is an important operation for imaging algorithms involving major cleaning loops of model refinement.
- *Division:* To improve calibration we might want to divide measurements by models to identify (linear) error terms.
- *Averaging:* Reducing data rates by averaging together suitably adjacent visibilities is easy. Obviously this requires visibility order to be chosen in a way that makes this possible.
- *Phase Rotation:* Changing the phase centre of visibilities involves multiplying the visibility with a baseline-position dependent phasor. Combined with averaging this can be used to produce facet visibilities.
- *Direct Fourier Transform:* In contrast to most other items on this list, this operation can become very costly once many sources need to be predicted. This means that it can be executed in a streaming manner, but runs a somewhat higher risk of falling behind if processing does not keep up fully.
- *Gridding:* To some point we can also grid as part of stream processing. However as section 3 will show, we cannot guarantee that all affected grids will be local to the node that loads the visibility, which means that we can only grid to temporary grids.

  Nevertheless, even such “pre-gridding” might be still be worthwhile to do in order to reduce data rates. After all, assuming a suitable sub-grid split and visibility order this

allows us to reduce data density to one "visibility" per grid cell, and allows us to possibly aggregate grid data from baselines that are close in uv-space[1]. We will touch on this again in subsection 3.8.

- *Degridding:* Dual to gridding, if a grid is available we can degrid predicted visibilities from it. This is the cheaper and less accurate alternative to direct Fourier transforms, but just as with gridding requires appropriate grid data to be made available.
- *Calibration:* Up to a certain point we can solve for calibration on streaming visibilities. Because of the limited window this will have to be very limited in scope, and it seems likely that we would only be able to calibrate out very clear signals this way. We will look more closely into this idea in subsection 4.3.

So in fact, quite a few of the important operations of radio astronomical pipelines work well with small data windows. The two main exceptions we had to make were de/gridding and calibration. The underlying reason is of course that by design grids, images and calibration problems correspond to large windows of visibility data. Therefore we need scatter and gather operations around the sketched stream processing. This leads to new working set problems, which we will investigate in the next sections.

3 Continuum Imaging

Processing as discussed so far is so "embarrassingly" parallel that it puts basically no restrictions on the order we process visibilities. This is in stark contrast to imaging pipelines, which impose strong locality requirements on loading and producing data. This means that we need to have a serious look into how we can distribute this processing under the assumptions derived so far. The hardest case here is the self-calibration pipeline ICAL, as in contrast to spectral pipelines it is meant to identify components and solve calibrations in relatively coarse frequency windows. This makes it a hard pipeline to distribute effectively. Furthermore, the requirement to image to the third null of the beam yields large images, which blows up the working set size significantly.

So clearly we need to consider imaging locality at least a secondary goal here. To be concrete, we need to check that the data distribution assumptions we just made about visibilities will not make it impossible to deal with the second most data-intensive data object in our pipeline: Raw images and and their underlying grids.

3.1 Sub-Band Split

Despite it being a relatively coarse split compared with spectral pipelines, the most low-hanging fruit here is still the frequency axis. After all, this is where visibility and image locality align: Simply by "pinning" workers to a band we reduce the amount of image information any single node has to care about. If visibility data is distributed accordingly, this is basically free[2].

[1] Thanks to Bram Veenboer for noting this!

[2] In fact, if we worked on sub-bands sequentially, we could even spill sub-band images to disk. This is actually exactly what we are going to do for the spectral pipelines considered in section 5. Yet for continuum pipelines we are actually somewhat short on "cheap" parallelism, so we do not consider that option here.

However, this is not enough by far – even just the image for one frequency sub-band is already very heavy. We can calculate:

$$\begin{aligned} N_{\text{tt}} N_{\text{pp}} M_{\text{image}} &= N_{\text{tt}} N_{\text{pp}} M_{\text{px}} N_{\text{pix,total}}^2 \\ &\approx 0.278\,\text{TB} && \text{for Low ICAL} \\ &\approx 14.23\,\text{TB} && \text{for Mid1 ICAL} \end{aligned}$$

Assuming $N_{\text{tt}} = 3$. This is clearly still too much for a node to aggregate locally. Unfortunately, in terms of the global working set this is actually already the end of the line: No matter how we turn the problem, every visibility eventually impacts this exact amount of grid/image data. Therefore streaming visibilities requires this amount to be held in memory somewhere.

While we cannot reduce the global working set, we can still reduce the working set *per node* by distributing the load. After all, at this point our distribution degree is just $N_{\text{subbands}} = 7$ (Low) or $N_{\text{subbands}} = 4$ (Mid1) respectively, which is by far not enough parallelism to use a large-scale compute facility effectively.

3.2 Cost of Distribution

So we require more distribution, both to split computational cost and to reduce the working set. Unfortunately, no other distribution option is quite as "embarrassingly parallel" as splits by sub-band. No matter what we do, we have to incur additional cost in some form. Those costs are, in rough order of badness:

Work: In some cases the split might not simply be about selecting or weighting visibilities, but might involve significant computational splitting/merging work. In that case we have to be careful that we do not introduce a new computational bottleneck into the pipeline.

Communication: We generally assume that visibilities will be distributed "optimally" across nodes before we start processing. However, for some distributions the same chunk of visibility data might still end up as a data dependency for different nodes. This requires us to introduce node communication. Depending on how many nodes take part in the distribution, we might even be forced to exchange data between compute islands.

Note that on paper, we might consider cutting down communication cost by duplicating visibility data in the buffer. However, this would directly increase both hot buffer capacity *and* bandwidth. Furthermore, in practice this would duplicate all visibility stream processing discussed before (including direct Fourier transforms and phase rotation). This would become prohibitively expensive pretty much immediately.

Working Set: Finally we might fall back to having copies of the grid/image that can be updated or predicted from independently. For the purpose of this analysis, this is going to be our last resort once we reach the limits of what our architecture can otherwise support.

3.3 Distribution Options

Fortunately, the Fourier transformation underlying imaging is a rather malleable method. As Table 1 shows, we can split along quite a number of axes:

- *Direction:* Imaging with a different phase centres still requires covering the whole uv-grid with visibilities, leading to an all-to-all communication pattern as shown in Figure 3.

Axis	Size	Extra Work	Communication	Working Set
Frequency (subband)	4-7	none	none	constant
Polarisation (separate)	2-4	none	none	constant
Direction (facet)	flexible	phase rotation	roughly constant	constant
Polarisation (cross)	2-4	none	linear	constant
Taylor terms	2-5	none	linear	constant
Frequency (rest)	flexible	none	none	linear
Time (snapshot)	3-36	none	none	linear
Baseline	flexible	imbalances?	none	linear?

Table 1: Continuum Imaging Distribution Options

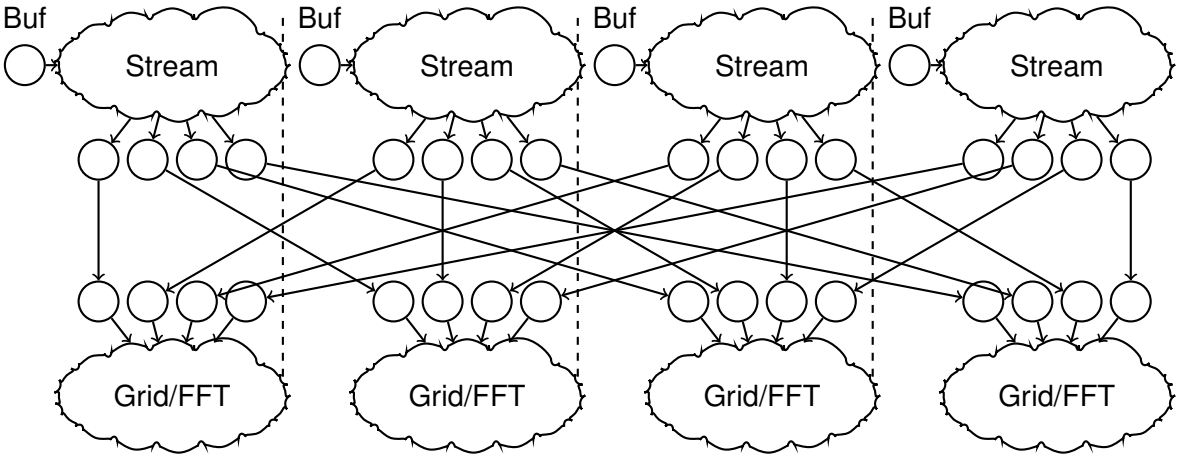

Figure 3: All-to-all Distribution Pattern of Facets, Polarisations and Taylor Terms

This might seem highly undesirable, but fortunately the reduced field of view per facet reduces sampling requirements in the uv-grid, which means that we can average visibilities more aggressively. This mostly offsets the fact that we need to produce visibility sets per direction: In Figure 3 every arrow only carries data roughly $\frac{1}{16}$ the size of the total visibility data. The amount of network transfer is effectively the same as if we were re-shuffling all visibility data between the nodes once.

However, the phase rotation and averaging operations required for producing visibilities for a facet field of view are not free: The obvious implementation would scale quadratically with the number of facets, which can easily introduce significant costs. There might be some optimisation potential here – such as employing FFTs or a direct divide-and-conquer approach – but there is always going to be cost.

- *Polarisation:* Without cross-polarisation terms, this would actually be another "embarrassingly" parallel distribution axis just like sub-bands. However, the assumption is that we need to account for those terms, which means that the image for any given polarisation would require the visibility data for *all* input polarisations.

  This introduces the same communication patterns as illustrated in Figure 3, yet in contrast to facets we have no easy in-built mechanism for data reduction. Therefore distributing in polarisation scales up the amount of network transfer in a linear fashion.

- *Taylor Terms:* Based on weighted visibilities, therefore needs all visibilities when distributed. This leads to the same kind of communication as distribution in polarisation.
- *Time:* Outside of fast imaging (which we do not consider here), no pipeline produces images split by time, so any distribution would increase working set size by preventing aggregation.

  We might still want to distribute along this axis simply because it naturally aligns with how visibilities are measured and snapshot imaging operates. While this will increase working set size, it provides a flexible way to scale node load without a lot of communication.
- *Baseline:* Dual to faceting – accumulate portions of the uv-grid. While this is a very natural choice for optimising visibility data locality, this makes balancing work harder, as visibility density varies a lot over the uv plane. Furthermore, in contrast to images uv-grids can not trivially be accumulated across projection changes (as with w-snapshots).

  Therefore this basically requires a full time split (or a second-stage re-distribution), which would seem to eliminate the benefit. We will however consider this topic again when considering sequential computation order in subsection 3.8.

3.4 Facetting Working Set

Clearly direction (faceting) is the most promising distribution axis here. If we follow parametric model (Bolton et al., 2016) recommendations for sizing, for Mid1 we can cover the field of view to the third null of Mid1 with 7×7 facets without having too pay too much for phase rotation. This also fits fairly well into the island size of $N_{\text{island}} = 56$ (Graser, 2016).

This would reduce the amount of image data every node has to hold to:

$$\begin{aligned} M_{\text{image/node,max}} &= \frac{M_{\text{image}} N_{\text{pp}} N_{\text{tt}}}{N_{\text{facet}}^2} \\ &\approx 8.17\,\text{GB} \quad \text{[Low ICAL]} \\ &\approx 290.5\,\text{GB} \quad \text{[Mid1 ICAL]} \end{aligned}$$

Which is not enough for SKA1 Mid, so we have to look further.

At this point we might be tempted to use local storage to off-load some of this working set. If we assume that we need to write a whole image out after every snapshot (for $t_{\text{snap}} = 600\,\text{s}$), this would result in the following write rates:

$$\begin{aligned} R_{\text{image}} &= \frac{N_{\text{major}} Q_{\text{speed}} M_{\text{image/node,max}}}{t_{\text{snap}}} \\ &\approx 0.204\,\text{GB/s} \quad \text{[Low ICAL]} \\ &\approx 7.263\,\text{GB/s} \quad \text{[Mid1 ICAL]} \end{aligned}$$

Unfortunately, the point where the working set becomes problematic coincides with the point where we go past the hot buffer read rate. As a high write rate is somewhat harder to achieve compared with a high read rate, we must take this seriously – spilling does not help us here.

3.5 Parallelism

If we assume that the cluster will have $N_{\text{node}} \approx 1500$ nodes with $N_{\text{subbands}} = 7$ (Low) and $N_{\text{subbands}} = 4$ (Mid1) this means we would want to aim for $N_{\text{node}} \approx 218$ (Low) or $N_{\text{node}} \approx 334$

(Mid1) respectively in order to have enough parallelism. This means that assuming a simple 7×7 facet split as recommended by the parametric model provides around 4-7 times less parallelism than we require.

Note that an alternative strategy here might be to reduce N_{node} by splitting compute resources between processing blocks (Nikolic and Bolton, 2016). This would require increasing hot buffer capacity, but not hot buffer bandwidth, as computation could take a longer time to finish. However, while this might introduce enough parallelism from the top level, it would not help with the working set problem: Every node would still have to hold all polarisations and Taylor terms for a facet. So this does not work well for pipelines involving high-resolution images such as SKA1 Mid.

3.6 Distributing Further

At this point, we have run out of "easy" choices. We might simply choose to use more facets, yet 14×14 or 19×19 facets would add significant phase rotation costs (roughly 1-2 Petaflops per second). So alternatively let us try splitting by another distribution axis mentioned in Table 1. There are only two classes of options left:

- Reduce working set per node at the expense of communication by increasing distribution in Taylor terms or polarisation.
- Keep communication constant at the expense of global working set size by distributing in time or frequency.

As two of the distribution axes each are the same for our purposes, let us say that we have a combined distribution degree of $N_{\text{dist,pp,tt}}$ in either polarisation and Taylor terms (or both), and a combined distribution degree of $N_{\text{dist,f,t}}$ in either time or frequency (or, again, both). If we want exactly enough parallelism to keep N_{node} nodes busy, we can calculate them from each other:

$$N_{\text{dist,f,t}} = \max\left\{1, \frac{N_{\text{node}}}{N_{\text{subbands}} N_{\text{facet}}^2 N_{\text{dist,pp,tt}}}\right\}$$

This allows us to scale the working set size per node quite a bit, potentially until the point where every node only has one facet worth of image data to keep track of for every node:

$$M_{\text{image/node}} = \max\left\{\frac{N_{\text{dist,f,t}} N_{\text{subbands}} N_{\text{pp}} N_{\text{tt}} M_{\text{image}}}{N_{\text{node}}}, M_{\text{facet}}\right\}$$

See the solid lines in Figure 4 for values depending on the value of $N_{\text{dist,pp,tt}}$ (and therefore $N_{\text{dist,f,t}}$). It is clear that we can reduce the working set per node significantly by distributing in polarisation and Taylor terms. Note that because of the different number of subbands we run into the lower working set limit at different points for Low and Mid1.

3.7 Communication

Reducing the working set like this is clearly very desirable. However, as noted before we will need more communication to make it happen. After all, now we need to copy the facet visibilities between nodes a few times. In fact, we can easily see that this requires communication with all nodes *except* those working on a different block in time and frequency, which means it works out as:

$$N_{\text{copy}} = \frac{N_{\text{node}}}{N_{\text{dist,f,t}} N_{\text{subbands}}} = \min\left\{\frac{N_{\text{node}}}{N_{\text{subbands}}}, N_{\text{facet}}^2 N_{\text{dist,pp,tt}}\right\}$$

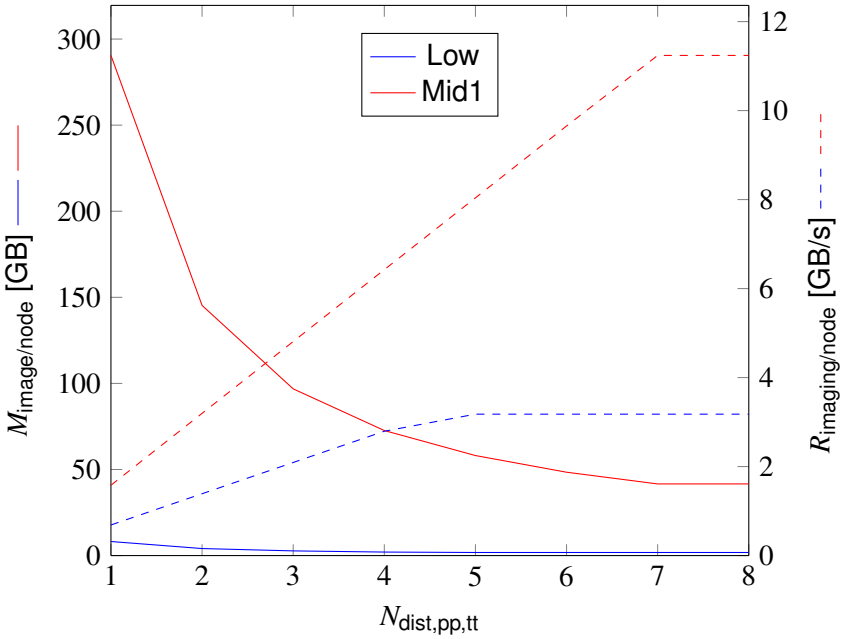

Figure 4: Imaging Working Set and Communication per Node (for 7×7 facets)

This means that if we have $R_{\text{vis,facet}}$ as the total averaged visibility rate for a facet, we can calculate the communication of every node as:

$$R_{\text{imaging/node}} = (N_{\text{copy}} - 1)\frac{Q_{\text{speed}} R_{\text{vis,facet}}}{N_{\text{node}}}$$

Results depending on the value of $N_{\text{dist,pp,tt}}$ are shown in Figure 4 as dashed lines. As should be expected, every increase of $N_{\text{dist,pp,tt}}$ corresponds to an increase of network I/O in the same order of magnitude as the visibility rate.

Especially note that this communication would be between N_{copy} nodes in an all-to-all fashion as illustrated back in Figure 3. This means that once the value of N_{copy} passes the number of nodes that a compute island contains (we assume $N_{\text{island}} = 56$) communication must happen between compute islands.

We can calculate the rate every island would have to emit in a fashion similar to $R_{\text{imaging/node}}$:

$$R_{\text{imaging/island}} = (N_{\text{copy}} - N_{\text{island}})\frac{N_{\text{island}} Q_{\text{speed}} R_{\text{vis,facet}}}{N_{\text{node}}}$$

As Figure 5 shows, once N_{copy} passes N_{island} this adds up rather quickly. This is traffic emitted per island, so for over 20 islands what we are seeing here is global traffic blow way past 10 TB/s.

However, note that traffic between each given pair of islands involved in the all-to-all is roughly constant independent of $N_{\text{dist,pp,tt}}$ and given by:

$$\begin{aligned} R_{\text{imaging/island2island}} &= \frac{N_{\text{island}}^2 Q_{\text{speed}} R_{\text{vis,facet}}}{N_{\text{node}}} \\ &\approx 46.72\,\text{GB/s} \quad \text{for SKA1 Low} \\ &\approx 94.23\,\text{GB/s} \quad \text{for SKA1 Mid1} \end{aligned}$$

So whether or not this is possible might depend exactly on whether we can sustain this sort of data rate between islands. It will likely take some careful hardware planning and experimentation to figure out the most workable compromise here.

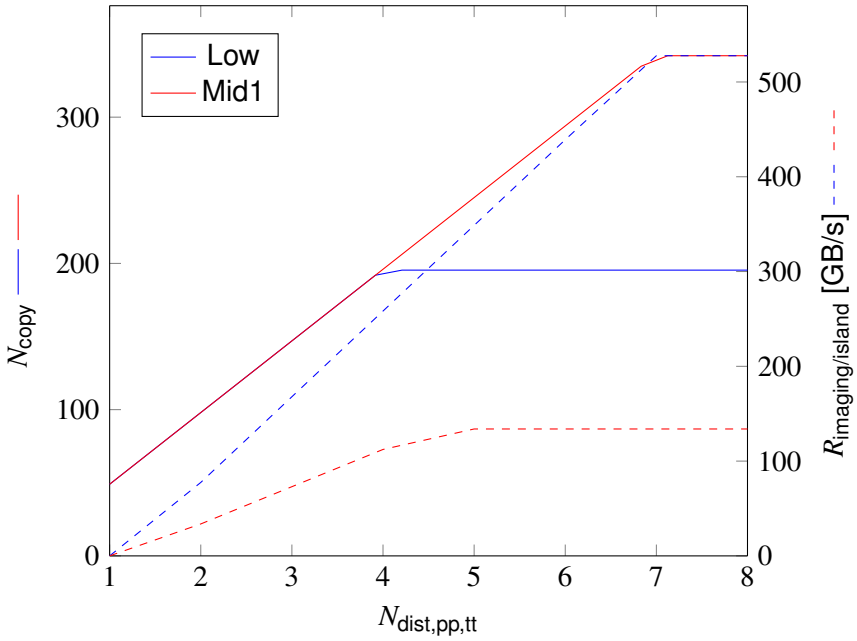

Figure 5: Imaging Island Communication (for 7×7 facets)

3.8 Streaming Order

We have covered how we could distribute the image data across nodes in order to minimise working set size and communication. However, to make streaming work as explained in subsection 2.3, we also need to show that we have a viable order in which we can process the visibilities without the working set blowing up further. Fortunately, sequential distribution has the significant advantage that we can aggregate in-place somewhat cheaply. We can split visibilities almost arbitrarily much in frequency, time and baseline without impacting our ability to work with the resulting chunks.

The main exception is snapshots: Clearly we would want to work in a way that not too many snapshots are worked on at the same time, as that would multiply up the amount of image/grid data that needs to be in memory at the same time. This means we will want to "softly" synchronise nodes collaborating on an imaging task: The grids for predict and imaging a certain snapshot should become available just as the first node requires them. Note that the overhead due to this is going to be lower the larger snapshots are.

Furthermore as noted in subsection 3.3, we might have to be quite picky with how we choose visibility order to optimise de/gridding. After all, w-stacking would become either memory inefficient or require many redundant FFTs unless we cluster visibilities that are close in uvw space together. This would even allow us to move gridding work on the order side of the communication, replacing phase-rotated visibilities with sub-grids.[3]. This is a very attractive design point: Even taking baseline-dependent visibility averaging into account, transferring gridded visibilities seems like it could decrease required communication by an order of magnitude. As illustrated in Figure 6, this suggests a very concrete way to distribute and stream visibilities: After initial chunking to some chosen granularity in baseline, time and

[3]Obvious choice would be to account for anti-aliasing, A-terms and fine w-terms on the streaming side. The node holding the image data would take care of coarse w-terms using w-stacking. Note this also has the nice property that most A-patterns are only required on very few nodes.

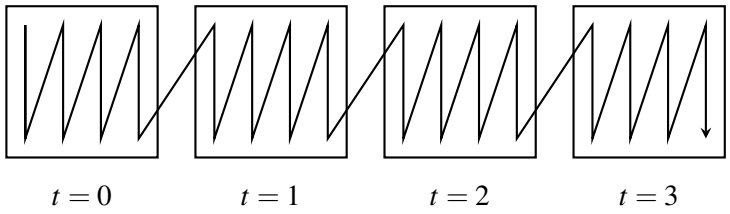

Figure 6: Sequential Work Order across Snapshot Grids

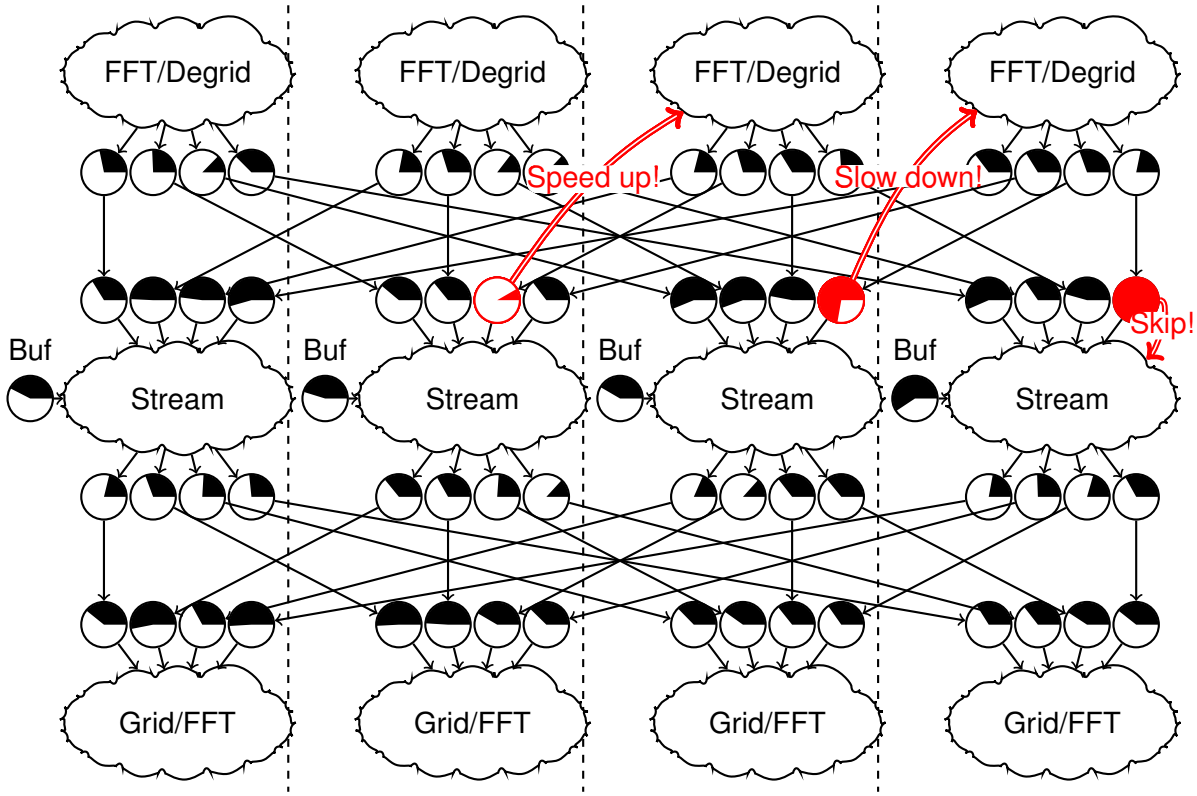

Figure 7: Back-pressure Throttling of Predict

frequency[4], we order visibilities first by snapshot, then by position in the $uv(w)$-grid. Then we would "stripe" the visibility chunks for this order into distributed storage, so that neighbouring visibilities will be loaded at roughly the same time.

Making the work order entirely deterministic makes the whole system very easy to describe in terms of streams: Facet prediction processes will stream visibilities (degridded or not) to the visibility processes in exactly the required order, and on the flip side nodes holding facet data will receive grid contributions back in the same predictable order. This especially makes it relatively straightforward to load-balance computation and communication. This is a critically important feature for SDP: Remember that for predict, we want to generate visibilities just as the measured visibilities get loaded from the buffer. Clearly the cluster collectively must be powerful enough to supply model visibilities at a higher rate than we can read visibilities from the buffer. Therefore there is the real risk that we might overwhelm visibility processing.

[4] A-kernel scopes are the obvious lower bound. Concrete granularity should depend on visibility density in uv-grid.

3.9 Implementation Sketch

The standard solution here seems to be to use back-pressure type mechanisms (Collins and Carloni, 2009) as shown in Figure 7 to throttle predict to match the buffer rate:

- Once we reach a high water mark on predicted visibilities, start throttling predict.
- When we reach a low water mark on predicted visibilities, reduce predict throttling.

Note that this balancing process would have to be "democratic", as every predict has multiple consumers potentially running at slightly different speeds. This means that it is impossible to truly prevent buffer over- or underflows. Fortunately though, most of the streaming visibility data is "non-precious", so we can actually exhaustively deal with all possible situations:

- If predicted visibilities overflow, this means we stop accepting predicted visibilities, then jump over buffer visibilities. This will allow the buffer and processing to catch up.
- If we actually run out of data on a buffer, the kernel will have to wait. This might cause other input buffers to overflow and skip data using the same mechanism.

This could be combined with additional load-balancing techniques, but it is likely that our data flows are predictable enough that this would be needed only rarely. In the end, the SDP should likely prioritise raw I/O at the prize of the risk of some data loss here.

4 Calibration

Outside of refining the sky model, the other primary reason for running the self-calibration pipeline is to find improved calibration solutions. This is a very complex topic, but the general goal is always to calibrate out a certain bias, generally per antenna / station:

- *Bandpass:* Fine calibration for sensitivity in specific frequencies
- *Gains:* Phase errors of the visibility
- *Pointing:* Errors in primary beam position
- *Ionosphere:* Various delays due to the beam passing through the ionosphere

While there are many different types of calibration, the principle seems to stay the same: We obtain a model for what we expect visibilities to be, which we then compare with the observed visibilities. We then proceed to tease apart the bias from other calibration effects or – worse – the signal. This is done by strategically comparing, grouping and averaging the measured and model visibilities until enough proof for systematic errors is found.

4.1 Challenges

Unfortunately, the practice of calibration is somewhat at odds with the restrictions on data flow as we have explained them so far. There are two main problems here:

Firstly, there is often significant overlap between calibrations. For example, gain errors only differ from ionospheric errors by how and how quickly we assume them to change in time, frequency and direction. Worse, even the same types of calibrations might have non-trivial interactions: As long as we have not obtained a precise calibration into the direction of a bright

source, the errors of that miscalculation might throw off all calibration on calibrators in the general vicinity.This leads to algorithms like LOFAR's facet calibration (Van Weeren et al., 2016), which calibrates facets entirely sequentially. This is a problem for our architecture, because as established in subsection 2.3 we can not actually "go back" very often – the system is sized for reading visibilities just about a few dozen times across its lifetime in offline processing.

Furthermore, for calibration to work effectively we must achieve as much signal-to-noise as we can. This means that we want to involve as many visibilities as possible: For high-quality bandpass calibration we would ideally want to take *all* visibility data that applies to a certain frequency range into account. Unfortunately, by nature different calibrations will want to slice data in entirely different ways: Just as bandpass calibration requires large time windows but small frequency windows, gain calibration would want small time windows, yet relatively large frequency windows.

4.2 Implementation Options

Viewed in isolation there is no "right" data distribution for visibility data: Every possible choice forces at least one calibration to perform a – possibly even global – redistribution of calibration data before we can solve it. This makes it hard to fit into a situation where we want to stream visibilities through under high memory pressure like with predict and invert.

This restricts our design space to two distinct algorithmic options:

1. Adopt the order the visibilities are streamed in, and therefore accept extremely limited data windows dictated by memory pressure. This forces calibration into the same working order as either imaging or ingest, which restricts the types and quality of calibration.
2. Commit to a data reorganisation with heavy aggregation in order to not exhaust memory. This also means large delays for actually obtaining and applying solutions, as collecting all required input data might take a full major loop.

We will look at these two options in detail in the next two sections.

4.3 Streaming Calibration

Let us first consider the option of calibrating within a given visibility stream. While such a calibration might have access to some "wisdom" from visibilities earlier in the stream, previous calibration runs or (non-guaranteed!) other calibration processes running in parallel – the only data window it is really guaranteed to have is what it happens to find in the streaming cache.

The good news is that such streaming calibration has unrestricted access to full-accuracy visibility data. This especially means that it could iterate and stack calibration methods on top of each other as much as the processing budget allows. This might be the appropriate way to implement calibration methods such as peeling (Intema et al., 2009) for sources that are bright enough to detect in any visibility snapshot of sufficient size (such as the A-team?).

A significant advantage of such sequential calibration is that we do not need to worry about calibrating out effects twice: We can apply the calibration right after it was solved, and do not have to worry about residual effects confusing other calibration solvers.

4.3.1 Alongside Imaging

Pairing streaming calibration with imaging might not look too problematic on first glance: We expect to have around 64 GB of data to work with, that should allow us to do something, right? However, if we take a closer look back at subsection 3.8, things become a bit more tricky.

We said that we would expect visibilities to be traversed by snapshot, then by region in uv-space. The former is good, as we would likely stream the types of calibration that are very localised in time anyway. However the latter is very bad, as calibration typically requires visibilities for all baselines, which by design are very spread out over the uv grid. If we assume that we only re-visit a certain region on the grid for every snapshot with $t_{\text{snap}} = 600\,\text{s}$, this means we need to hold an entire snapshot worth of visibilities in working memory for:

$$t_{\text{work,snap}} = \frac{t_{\text{snap}}}{Q_{\text{speed}} N_{\text{major}}} \approx 40\,\text{s}$$

which compares unfavourably with $t_{\text{work}} < 3\,\text{s}$, the time we gave visibilities to stay memory-resident in subsection 2.2. We can easily calculate that in order to give calibration a window large enough to cover a snapshot we would have to hold:

$$\begin{aligned} M_{\text{snap/node}} &= \frac{Q_{\text{speed}} N_{\text{major}} N_{\text{pp}} R_{\text{vis}} t_{\text{snap}}}{N_{\text{node}}} \\ &\approx 183.60\,\text{GB} \quad \text{[SKA1 Low]} \\ &\approx 175.29\,\text{GB} \quad \text{[SKA1 Mid1]} \end{aligned}$$

Note that in contrast to the calculation in subsection 2.2 we now assume that we do not need to hold model visibilities for this long, which is why this looks less crass. This mostly rules out distributed predict, and therefore effectively limits us to model visibilities generated by just-in-time direct Fourier transforms – which seems appropriate in this case.

However, this is still a very significant amount of data, especially considering that this would have to be added on top of all previously mentioned visibility buffering. We might be able to improve this number by choosing smaller snapshots, or – equivalently – make multiple passes through the grid (both would introduce more FFT cost). Also note that this would restrict how we could stripe visibility data among nodes, as we need to make sure that every one has a complete set of baseline visibilities to solve calibration for a certain slot in time and frequency.

These are quite thorny requirements – it should illustrate why calibration is such a problem for our architecture. However, it might still be worth the price: For SKA1 Low we have relatively small imaging working sets, and exceptionally tough calibration problems. So setting aside 200 GB of visibility re-ordering buffer per node for calibration should not be dismissed.

4.3.2 Pre-Processing

If we take a broader look of the life cycle of visibilities, during continuum imaging might well be the worst time to try to do streaming calibration. Remember that visibilities are also naturally ingested in a streaming fashion – in an order that is much less problematic for calibration solving. After all, we need to be able run the real-time calibration pipeline (RCAL)!

This heavily suggests that calibrations that do not need distributed predict should be solved *before* we even enter the self-calibration pipeline. We have a good number of options for doing that within the SDP architecture:

- The obvious first suspects would be **Receive** and **RCAL** – we already need to solve some calibration right away. As these pipelines will primarily serve real-time requirements, there is however no guarantee that this will be of high enough quality for ICAL.
- **Real-time stream calibration:** We might introduce a proper real-time calibration solving pipeline (variant/extension of RCAL) that prepares calibration solutions for ICAL. However, note that at this stage we would have no way to look ahead in measurement data.

- **Off-line stream calibration:** Once the data has been written into the (cold) buffer, we might want to re-read the data for refining calibration. In contrast to real-time pipelines, this has the advantage that it could take "wisdom" from adjacent calibrator observation runs into account.

  Especially note that this could work interactively: We could use this process to selectively investigate and re-calibrate suspect parts of the observation. This might be an essential tool in probing and refining the quality of the data ahead of the expensive ICAL run.

- **Staging stream calibration:** Before we start the ICAL pipeline, we will presumably transfer the data to the "hot" high-performance buffer space, re-order it with imaging efficiency in mind.

  This is the last time we touch the data in its original order, and at no extra cost architecturally. This should be seen as a prime opportunity to apply the last touches to streaming calibration solutions.

In summary: We should be able to go a long way towards exhausting the potential of stream calibration way ahead of entering ICAL. This should sit well with the types of calibration that we would want to apply in a streaming fashion. After all, obtaining good calibration solutions for bright sources is going to be an essential first step for any radio astronomy pipeline. Therefore getting it out of the way early not only makes our life easier in terms of data distribution, but also reduces the chance that we end up wasting resources on junk data.

4.4 Global Calibration

Instead of trying to stream calibration, we might instead go into the opposite direction and parallelise calibration solving: Instead of generating a solution right when the visibilities are loaded, we "extract" the calibration problem from the data as shown in Figure 2, and solve it once it has been accumulated in its entirety.

De-coupling calibration solving from the main visibility stream has significant advantages: We can depend on basically arbitrary visibility data, and have until the next major loop to find our solutions. This provides ample time to cooperate with other nodes around the cluster and find a "global" calibration solution, for example using an algorithm like `SageCal-Co` (Patil et al., 2017). It is especially quite "embarrassingly" parallel, with over half a million mostly (!) independent problems to solve for SKA1 Low, as Table 2 shows. This should make it relatively easy to load-balance effectively using classic graph-based scheduling.

And while this means that we cannot work on – say – directions sequentially, this design still leaves us quite a bit of room to manoeuvre algorithmically: As Figure 8 illustrates, we can still subtract out effects from outside the facet field of view, then solve calibration for the residual visibilities. This should be roughly equivalent to the data flow sketched in Van Weeren et al. (2016), except we solve all directions at the same time, and can not iterate as much (as presumably $N_{\text{facet}} \gg N_{\text{major}}$). Note the complex data flow between Predict and Subtract: This is only possible because we assume the data streams to end up on common nodes already.

4.5 Calibration Data

To make global calibration work, we have a number of working set considerations to get out of the way. After all, we said that we want to solve millions of calibration problems over the course of a major loop. This is great for parallelism, but bad for the working set: The other side of the coin is that we now need to account for the combined working set for all of these calibrations.

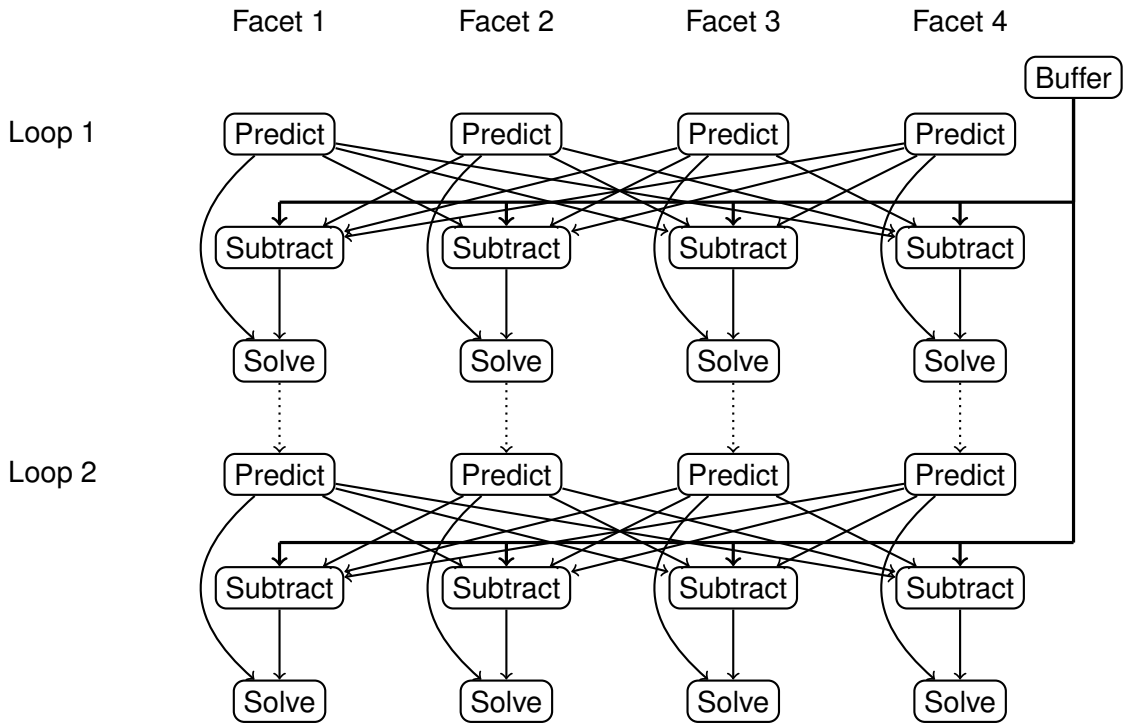

Figure 8: Parallel Facet Calibration

Unfortunately, the combined working set of all calibration problems is basically the entire observation plus model visibilities, and as established this is more data than we could possibly hold. However, there is good reason to expect calibration solvers to have good options for input aggregation. After all, calibration is by nature a very strong data reduction operation. We can find many examples for calibration solvers using various strategies to reduce their input data:

- Averaging input visibilities is a common strategy when consider specific sky directions, such as with peeling (Intema et al., 2009).
- Measured and model visibilities can also be used to fomulate normal equations, which can be averaged (Ord, 2016).
- Finally, as finding calibration solutions is quite cheap we can also solve calibrations separately per data window, then average the solutions as in Patil et al. (2017). Note however that to effectively reduce the working set this must be a one-way street, we can only re-visit the calibration input data again for the next major loop.

These approaches could presumably even be combined. Therefore it seems likely that we would be able to identify some effective aggregation scheme for a significant number of useful calibration algorithms . Let us therefore assume that an individual global calibration "problem" resulting in a calibration solution per antenna can be solved from 2 visibilities (measurement + model) worth of data per baseline.

To calculate the working set, we would have to assume this data to be resident throughout the major loop. This is because it might take a lot of time to collect all visibilities, plus the calibration algorithm itself will typically iteratively re-visit the input data until a stable solution

[5] SKA1 Mid does not require ionospheric calibration. We leave it in as a place-holder for pointing calibration.

Calibration	$N_{\text{dir,cal}}$	$N_{\text{f,cal}}$	t_{cal}	# Low	# Mid1	Data Low	Data Mid1
Gains	1	N_{subbands}	1 s	151 200	86 400	1.27 GB	0.11 GB
Bandpass	1	500	3600 s	3 000	3 000	0.03 GB	0.01 GB
Ionosphere[5]	30	N_{subbands}	10 s	453 600	8 640	3.80 GB	0.01 GB

Table 2: Calibration Windows, Count and Working Set per Node

	$N_{\text{cal/snap}}$		$M_{\text{cal/snap}}$		R_{cal}	
Calibration	Low	Mid1	Low	Mid1	Low	Mid1
Gains	600	600	7.53 GB	1.11 GB	0.188 GB/s	0.028 GB/s
Bandpass	72	125	0.90 GB	0.23 GB	0.022 GB/s	0.006 GB/s
Ionosphere	1 800	60	22.61 GB	0.11 GB	0.565 GB/s	0.003 GB/s

Table 3: Calibration Aggregation Count, Working Set and Data Rate per Node

is obtained. Calibration solvers might obviously have other working set requirements, but it seems like a good starting point. Fortunately as shown on the right side of Table 2, calibration data is geberally averaged quite aggressively. We can therefore calculate the working set as:

$$M_{\text{input,cal}} = 2M_{\text{vis}}N_{\text{pp}}N_{\text{baseline}}N_{\text{dir,cal}}N_{\text{f,cal}}\frac{t_{\text{obs}}}{t_{\text{cal}}}$$

Note that in contrast to imaging, calibration data will grow depending on how long the observation is – the values in the middle columns of Table 2 have been calculated for $t_{\text{obs}} = 6\,\text{h}$.

This illustrates that on paper it should be entirely possible for the cluster to hold all information required in memory at all times: The 5.1 GB needed for SKA1 Low per node are almost irrelevant in the grand scheme of things. Especially note that in contrast to image data, calibration solving will likely have long “stale” period – such as while waiting for the calibration data aggregation to complete or after convergence has been achieved – where we could spill the data to disk or remove it from memory entirely.

4.6 Aggregation

However, this working set calculation is only valid once the calibration input has been fully aggregated, which as established previously is the main problem here. We can not wait with aggregation until all data has been collected on a common node: That could easily overwhelm the network as well as the working memory of the node in question.

Therefore let us assume that the calibration input aggregation can be done in stages: Given a number of aggregated measured and model calibration data chunks we should be able to determine the combined calibration problem representation corresponding to the combined underlying visibility data sets. This allows us to put calibration input aggregation as a tree reduction, which gives us powerful tools for effective distribution (Dean and Ghemawat, 2008).

A natural first aggregation stage is snapshot $\times$ sub-band granularity on every node. This works well because the bulk of calibration data (gains and ionosphere) are fairly local in time. Furthermore, as explained in subsection 3.8 we would likely work on snapshots sequentially. This means that at every point a node only needs to hold the calibration data corresponding to the current snapshot in memory.

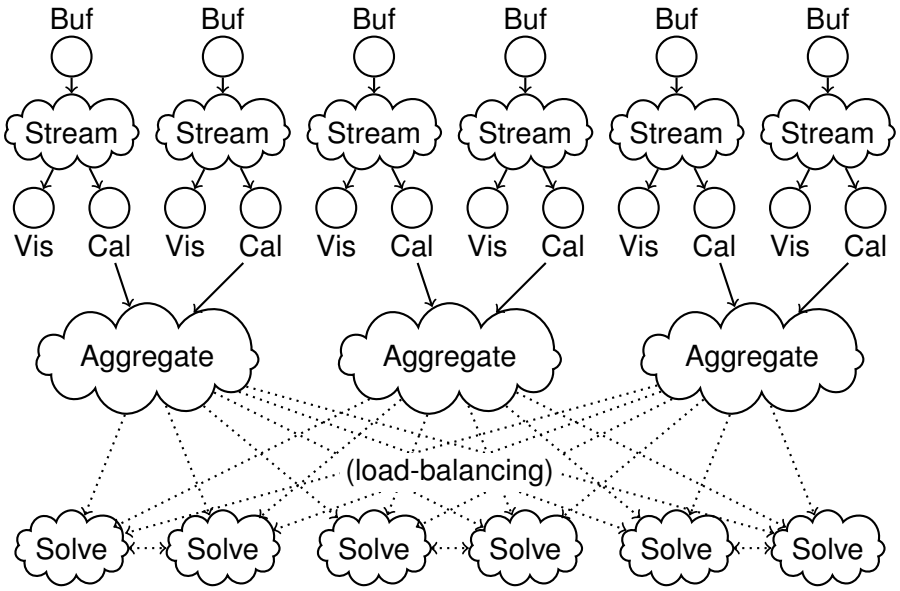

Figure 9: Calibration Input Aggregation and Re-Distribution Pattern across Islands

We can calculate that for a given calibration, a snapshot would overlap the following number of calibration problems:

$$N_{\text{cal/snap}} = N_{\text{dir,cal}} \max\{\frac{N_{\text{f,cal}}}{N_{\text{subbands}}}, 1\} \max\{\frac{t_{\text{snap}}}{t_{\text{cal}}}, 1\}$$

As Table 3 shows, this adds up to a couple thousand calibration problems per snapshot. Now, let us assume a maximally bad visibility distribution, where each node makes some contribution to every baseline of every single calibration problem overlapping the snapshot. Then we could calculate the working set size for visibility aggregation over snapshots as:

$$M_{\text{cal/snap}} = 2 M_{\text{vis}} N_{\text{pp}} N_{\text{baseline}} N_{\text{cal/snap}}$$

This results in a working set of about 30 GB per node for SKA1 Low. This is a rather conservative estimate, and there are obvious ways for improving it (such as aggregating only across partial snapshots), so this is a very reasonable number.

4.7 Communication

Assuming we have aggregated calibration input data for one snapshot, we still need to bring the data from all individual nodes into one place. Let us assume that each node needs to transfer the whole calibration input data set over the network each time it finishes processing a snapshot:

$$R_{\text{cal}} = \frac{Q_{\text{speed}} N_{\text{major}} M_{\text{cal/snap}}}{t_{\text{snap}}}$$

As shown in Table 3, this yields relatively minor data rates of up to 700 MB/s per node. Especially note that we can exploit the network infrastructure in a straight-forward way here by introducing another level of tree reduction at the island level. This is especially effective because as we argued in subsection 3.8, nodes would likely work on snapshots in loose synchronisation, which

makes it attractive to gather contributions before making the exchange with other islands as illustrated in the upper half of Figure 9. This means that for the last stage of tree reduction every island would have to emit just R_{cal} worth of calibration data.

4.8 Distribution

While it is a good idea to aggregate calibration input data for reducing data, we have to keep in mind that in the end we intend to solve the calibration problems in a distributed fashion. This means that conceptually we are not actually talking about a single global tree reduction, but about a lot of separate tree reductions, culminating on different cluster nodes. This is especially important to consider because calibration is much less predictable architecturally: Solving strategies might vary wildly by calibration type, and might especially introduce distribution and hard-to-predict performance characteristics of their own. Clearly we want to be very careful with how we schedule calibration solving!

Fortunately, the island aggregation nodes are also a natural point to schedule calibration. As the lower half of Figure 9 shows, those nodes have relatively free reign in spreading calibration solving across the available nodes. And as Table 3 showed, even at the finest possible calibration problem level we would need to schedule and distribute just a few thousand calibration problems per snapshot (so roughly every 40 seconds). Even for a very distributed load balancing process, this seems unlikely to become a problem.

5 Spectral Pipelines

So far we have mostly focused on the continuum self-calibration pipeline ICAL. However, clearly this is not the only pipeline configuration SDP needs to run – it just happens to be the most important one, and most demanding in terms of frequency ranges and field of view to be considered. At the same time, most of our arguments apply equally well to other continuum imaging and coarse spectral pipelines. If we set N_{subbands} to the required number of output frequency channels[6] our calculations work out the same way as they did before.

However, the one significant exception is fine spectral pipelines. Way back in subsection 3.1 we identified the top-level frequency split as the "easiest" way we could distribute imaging, and had to find further sources of distribution to get enough parallelism and distribute the working set. For fine spectral pipelines, the situation is somewhat different: The general working set problems stay the same, but now we have *too much* parallelism for the cluster.

5.1 Working Set

We need to adjust our treatment of imaging to allow us to absorb surplus parallelism. If we look back at subsection 3.6 we quickly find that for spectral pipelines with $N_{\text{subbands}} > N_{\text{node}}$ we have:

$$N_{\text{dist,f,t}} N_{\text{dist,pp,tt}} N_{\text{subbands}} N_{\text{facet}}^2 \gg N_{\text{node}}$$

which means that we are forced to partly sequentialise work on output frequency channels:

$$N_{\text{seq}} = \max \left\{ 1, \frac{N_{\text{dist,pp,tt}} N_{\text{subbands}} N_{\text{facet}}^2}{N_{\text{node}}} \right\}$$

[6] Note we are stretching the meaning of "subbands" a bit, as for our purposes the pixel resolution does not matter. The parametric model uses $N_{\text{f,out}}$ instead to differentiate these.

For the definition of N_{copy} this means replacing $N_{\text{node}} \to N_{\text{seq}} N_{\text{node}}$:

$$N_{\text{copy}} = \frac{N_{\text{seq}} N_{\text{node}}}{N_{\text{dist,f,t}} N_{\text{subbands}}} = \min\left\{ \frac{N_{\text{seq}} N_{\text{node}}}{N_{\text{subbands}}}, N_{\text{facet}}^2 N_{\text{dist,pp,tt}} \right\}$$

Which is already enough to yield a consistent picture of imaging.

DPrepC pipelines do not need to image the field of view out to the third null, which means that we have less image data in general. This means that we get relatively good working sets even with just $N_{\text{facet}} = 7$ and $N_{\text{dist,pp,tt}} = 1$:

$$\begin{aligned} M_{\text{image/node}} &\approx 1.12\,\text{GB} \quad \text{for SKA1 Low} \\ &\approx 39.85\,\text{GB} \quad \text{for SKA1 Mid1} \end{aligned}$$

5.2 Communication

However, things look a bit different if we consider data rates, especially for the finest spectral resolutions. As parametrised in the parametric model, DPrepC is meant to produce an image cube with the full spectral resolution of $N_{\text{subbands}} \approx 65\,000$ input frequency channels.

Note that this configuration is quite extreme to the point that it seems highly unlikely to ever be run as modelled – even with the smaller field of view this would result in an output image cube of size 0.83 PB (SKA1 Low) and 29.62 PB (SKA1 Mid) respectively. For comparison: an observation yields around 2 PB input data per hour. So to give a better idea of what this would look like in reality, we will give results not just for DPrepC, but also for a more reduced DPrepC' configuration with $N_{\text{subbands}} = 3000$. This is the maximum spectral resolution that the parametric model currently actually lists a scientific use case for.

It might be slightly surprising because we are using the same input visibilities as ICAL, but for DPrepC we already hit a problem with visibility data rates. Sequentialising the treatment of output frequency channels is not a problem here, as the visibility data is simply split up between (now sequential) output frequency channels, and striped accordingly across nodes. However, what does matter is that by considering frequency channels independently we partly lose the ability to coalesce visibilities. This means that even with just $N_{\text{dist,pp,tt}} = 1$ and therefore $N_{\text{copy}} = N_{\text{facet}}^2 = 49$, if we re-evaluate the formula from subsection 3.7 for spectral configurations, the visibility rate for DPrepC is a lot higher:

$$\begin{aligned} R_{\text{imaging/node}} &= (N_{\text{copy}} - 1) \frac{Q_{\text{speed}} R_{\text{vis,facet}}}{N_{\text{node}}} \\ &= 23.2\,\text{GB/s} \quad \text{for Low DPrepC} \\ &= 8.4\,\text{GB/s} \quad \text{for Mid1 DPrepC} \\ &= 1.16\,\text{GB/s} \quad \text{for Low DPrepC'} \\ &= 1.03\,\text{GB/s} \quad \text{for Low DPrepC'} \end{aligned}$$

It is especially striking how much this affects SKA1 Low, where we get a big rate increase for the finest spectral configuration due to the high number of baselines. However note that we are not too constrained by the working set for SKA1 Low – so we might be able to just set $N_{\text{facet}} = 1$, which would eliminate this communication entirely. Furthermore, comparison with DPreC' shows that this effect reduces quickly as long as we permit some degree of coalescing.

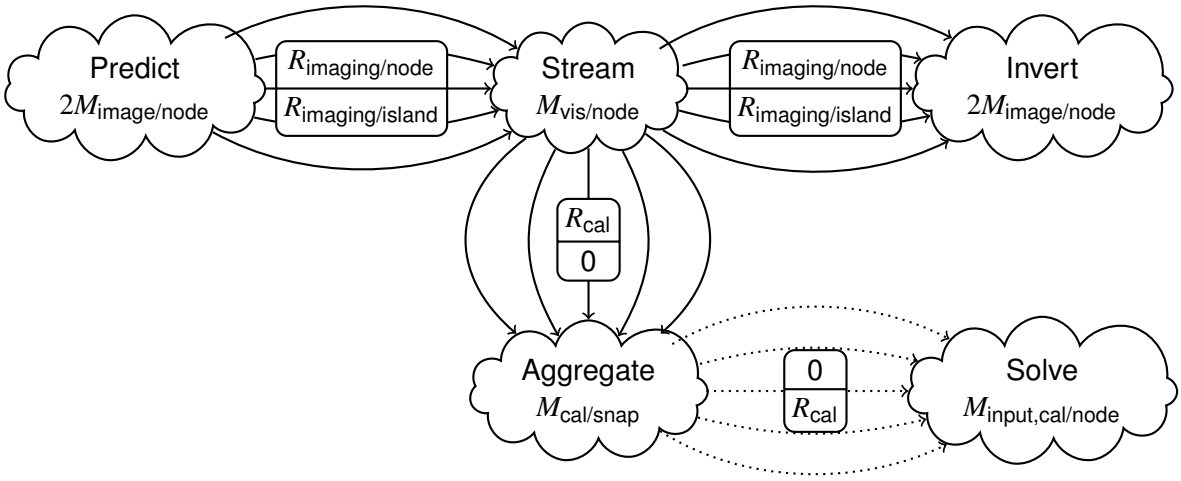

Figure 10: Illustration of Working Sets and Node/Island Rates

5.3 Data Products

Finally we have to touch on a topic that is entirely specific to spectral pipelines: The production rate of image data. After all, while working on frequencies in sequence does not increase visibility data rates, it will increase the rate at which we produce images.

Remember that in subsection 3.4 we pondered whether it might be worth it to "spill" the image data from individual snapshots to disk. We can modify the formula by introducing N_{seq} and using t_{obs} as the time basis to arrive at:

$$
\begin{aligned}
R_{\text{image,out}} = \frac{N_{\text{major}} N_{\text{seq}} Q_{\text{speed}} M_{\text{image/node}}}{t_{\text{obs}}} \\
&\approx 1.973\,\text{GB/s} \quad \text{for Low DPrepC} \\
&\approx 80.00\,\text{GB/s} \quad \text{for Mid1 DPrepC} \\
&\approx 0.090\,\text{GB/s} \quad \text{for Low DPrepC'} \\
&\approx 3.662\,\text{GB/s} \quad \text{for Mid1 DPrepC'}
\end{aligned}
$$

In contrast to ICAL, DPrepC *must* write out this data, as it is the primary output of the imaging pipeline. The rates are exceptionally high, which is not entirely surprising given that we that Mid1 DPrepC produces more outputs than it has visibility inputs. Therefore with the possible exception of Low DPrepC' we should expect spectral pipelines to be limited by the buffer write rate in the final data product generation step.

6 Conclusion

In this memo we have decomposed the SDP imaging and calibration pipelines into 5 distributed stages as shown in Figure 10. Each of those stages requires a certain working set and data re-organisation in order to achieve the required data locality:

- **Stream:** We put a visibility streaming process at the centre of the distribution scheme. Theoretically the working set should be very low, but we will need significant visibility streaming cache space $M_{\text{vis/node}}$ to ensure efficiency.

- **Predict + Invert:** Symmetrical in their requirements, these stages will need to hold image/grid data to provide/add (possibly gridded) visibilities from/to. Because of reprojection these stages will likely have to hold both a grid and an image each at every point, so it contributes $4M_{\text{image/node}}$ to the working set. Furthermore, re-distributing visibilities towards the visibility stream process will require a data rate of $2R_{\text{imaging/node}}$ per node.
- **Aggregate + Solve:** We dedicate two stages of data reduction to calibration to ensure flexibility in the types of calibration we can support. Aggregating calibration across a snapshot requires a node to worst-case hold $M_{\text{cal/snap}}$ calibration input data, with the off-node communication contributing R_{cal} to the overall network traffic.

As it stands, imaging is generally the most demanding on our architecture. For calibration we have the options of solving it ahead of time, trying to fit it in with imaging, or solving it globally. Each method has their own drawbacks, but neither seems to have a big impact on our architecture.

6.1 Data Flow

For this memo we have mostly focused on working sets and communication, but along the way we have also gathered a few insights about possible implementation choices. We have come across three characteristic types of data flow:

- **Grid Data Flow:** Possibly gridded visibilities need to be transferred from and to nodes holding the image/grid data. This means shuffling a lot of data between a lot of nodes (possibly across islands) in a predictable all-to-all pattern as illustrated in Figure 3.

  The principal concern here is to reach sufficient I/O performance (storage and network) to feed the data streams without running out of memory. As sketched in Figure 7, this might involve careful throttling of individual processes. Fortunately, this data flow will look very similar between pipelines, with only minor changes in – say – the number of facets. The processing done with this data would be obviously I/O bound, which suggests implementation on host CPUs.

- **Visibility Stream Data Flow:** Once measurement and model visibilities have been brought together, we enter a very different stage: Now we need to perform actual heavy processing. After all, if we move gridding and degridding to the stream stage as advocated in subsection 3.8, this represents roughly 90% of the predicted SDP flop cost!

  At the same time, we will have to implement fairly complex data dependency networks to serve different calibration types as noted in subsection 4.4. It is therefore absolutely crucial that we implement this stage only using operations that we can stream (have work with small chunks), like the ones listed in subsection 2.4. Fortunately, once we do this we might be able to reach high levels of memory locality and therefore operational intensity.

  Therefore we end up with quite a unique situation: We want high performance, which suggests we should target high-performance accelerators. At the same time, we want modifiability, which suggests usage of high-level execution frameworks. The good news is that we have eliminated most requirements for communication between nodes at this stage, so this might not actually be too hard.

- **Global Calibration Data Flow:** Finally, one of the outputs of the stream data flow is going to be calibration problems, which need to be aggregated and solved. Under the assumptions from subsection 4.6 this boils down to a robust Map/Reduce (Dean and

Band	Pipeline	$M_{\text{vis/node}}$	$M_{\text{image/node}}$ $R_{\text{imaging/node}}$ $R_{\text{imaging/island}}$	$M_{\text{cal/snap}}$ R_{cal}	$M_{\text{input,cal/node}}$ R_{cal}	M_{node} R_{node} R_{island}
Low	ICAL	64 GB?	2.04 GB 2.79 GB/s 112.35 GB/s	31.04 GB 0.78 GB/s	4.89 GB 0.78 GB/s	108.15 GB 6.36 GB/s 225.45 GB/s
Mid1	ICAL	64 GB?	72.63 GB 6.43 GB/s 258.55 GB/s	1.45 GB 0.06 GB/s	0.13 GB 0.06 GB/s	356.15 GB 12.95 GB/s 517.50 GB/s
Mid2	ICAL	64 GB?	63.39 GB 3.49 GB/s 140.45 GB/s	1.53 GB 0.04 GB/s	0.10 GB 0.04 GB/s	319.25 GB 7.02 GB/s 280.85 GB/s
Mid5A	ICAL	64 GB?	64.72 GB 2.46 GB/s 140.45 GB/s	1.69 GB 0.04 GB/s	0.07 GB 0.04 GB/s	324.65 GB 4.96 GB/s 197.65 GB/s
Mid5B	ICAL	64 GB?	64.72 GB 1.12 GB/s 45.15 GB/s	2.15 GB 0.05 GB/s	0.04 GB 0.05 GB/s	316.35 GB 2.31 GB/s 90.25 GB/s
Mid5C	ICAL	64 GB?	68.48 GB 4.22 GB/s 169.95 GB/s	1.53 GB 0.04 GB/s	0.10 GB 0.04 GB/s	339.55 GB 8.49 GB/s 339.85 GB/s

Table 4: Overview of Pipeline Working Sets and Node/Island Rates

Ghemawat, 2008) type of tree reduction. Computational cost can be expected to be relatively modest, but we have to expect calibration to evolve relatively quickly and display hard-to-predict performance characteristics.

This paints a pretty clear picture: There is not much to be gained here from chasing high performance for I/O or compute, instead most of the challenges will likely arise in terms of modifiability and load balancing. Therefore it seems likely that once the appropriate aggregations have been implemented systematically, calibration would be fairly well suited for general-purpose distributed execution frameworks.

6.2 Summary

As a final overview, Table 4 shows working sets and rates for different pipelines. All of these values were calculated with $N_{\text{facet}} = 7$ and $N_{\text{dist,pp,tt}} = 4$ to make it easier to compare. Note that the Mid1 ICAL configuration still looks the most problematic overall, showing both a big working set and a lot of inter-island traffic. However, for the moment the Science Data Processor architecture seems roughly on track.

List of symbols

$M_{\text{cal/snap}}$	Calibration problem data extracted from a snapshot subband
M_{facet}	Size of an image facet
M_{image}	Size of a full image (all facets)

$M_{\text{image/node}}$	Image data that needs to be held per node
$M_{\text{image/node,max}}$	Image data to hold per node for worst configuration
$M_{\text{image/node,min}}$	Image data to hold per node for best configuration
$M_{\text{input,cal}}$	Size of calibration input data
$M_{\text{input,cal/node}}$	Size of calibration input data per node (after distribution)
M_{node}	Working set per node
M_{px}	Byte size of a pixel
$M_{\text{snap/node}}$	Size of a snapshot subband per node working on it
M_{vis}	Byte size of a visibility
$M_{\text{vis/node}}$	Visibility data streaming cache per node
N_{baseline}	Number of baselines of a telescope
N_{copy}	Distribution degree of visibilities for predict/imaging
$N_{\text{cal/snap}}$	Number of calibrations depending on a snapshot subband
$N_{\text{dir,cal}}$	Number of directions to calibrate into
$N_{\text{dist,f,t}}$	Distribution degree in frequency and/or time
$N_{\text{dist,pp,tt}}$	Distribution degree in polarisations and/or Taylor terms
N_{facet}	Number of facets in horizontal/vertical direction
$N_{\text{f,cal}}$	Number of frequency channels to calibrate
N_{island}	Number of nodes in an island
N_{major}	Number of major loops (passes through data for imaging + calibration)
N_{node}	Number of nodes in cluster
$N_{\text{pix,total}}$	Number of pixels on a side of the final image (all facets)
N_{pp}	Number of polarisation parameters
N_{predict}	Number of model visibility sets needed
N_{seq}	Sequential distribution degree (used for fine spectral pipelines)
N_{subbands}	Number of subbands. For this memo, the number of output frequency channels for imaging.
N_{tt}	Number of Taylor terms
Q_{speed}	Factor between observation and processing time for a pipeline
R_{cal}	Calibration input data rate after subband snapshot aggregation
R_{image}	Snapshot image data production rate
$R_{\text{image,out}}$	Output image data production rate
$R_{\text{imaging/island}}$	Island input and output rate for distribution of imaging data
$R_{\text{imaging/island2island}}$	Pairwise island input and output rate for imaging data
$R_{\text{imaging/node}}$	Node input and output rate for distribution of imaging data
$R_{\text{input,cal}}$	Calibration input data production rate
R_{island}	Island input and output communication data rate
R_{node}	Node input and output communication data rate
$R_{\text{predict/node}}$	Model visibility production rate
R_{vis}	Visibility ingest rate
$R_{\text{vis,facet}}$	Visibility rate for a facet after coalescing
$R_{\text{vis/node}}$	Visibility read rate per node
t_{cal}	Time slot size for calibration
t_{chunk}	Chunk size for stream processing
t_{obs}	Observation length
t_{snap}	Snapshot length. Considered a constant in this memo.
t_{work}	Time allowed to work on streaming visibilities
$t_{\text{work,snap}}$	Time needed to work on a snapshot

List of Figures

List of Tables

References

Bolton, R. et al.: Parametric models of SDP compute requirements, Tech. Rep. SKA-TEL-SDP-0000040, SDP Consortium, dated 2015-03-24, 2016.

Collins, R. L. and Carloni, L. P.: Flexible filters: load balancing through backpressure for stream programs, in: Proceedings of the seventh ACM international conference on Embedded software, pp. 205–214, ACM, 2009.

Cornwell, T., Voronkov, M., and Humphreys, B.: Wide field imaging for the Square Kilometre Array, in: SPIE Optical Engineering+ Applications, pp. 85 000L–85 000L, International Society for Optics and Photonics, 2012.

Culler, D. E. and Arvind: Resource Requirements of Dataflow Programs, SIGARCH Comput. Archit. News, 16, 141–150, doi:10.1145/633625.52417, http://doi.acm.org/10.1145/633625.52417, 1988.

Dean, J. and Ghemawat, S.: MapReduce: simplified data processing on large clusters, Communications of the ACM, 51, 107–113, 2008.

Graser, F.: SDP Cost Model, Tech. Rep. SKA-TEL-SDP-0000043, SKA SDP Consortium, revision 02E, 2016.

Intema, H., Van der Tol, S., Cotton, W., Cohen, A., Van Bemmel, I., and Röttgering, H.: Ionospheric calibration of low frequency radio interferometric observations using the peeling scheme-I. Method description and first results, Astronomy & Astrophysics, 501, 1185–1205, 2009.

Nikolic, B. and Bolton, R.: A potential alternative architecture for the SDP: Large Visibility Buffer and Observation-based Task Parallelism, Tech. rep., SDP Consortium, https://confluence.ska-sdp.org/x/FoAuCw, dated 2016-04-25, 2016.

Nikolic, B., Dijkema, T.-J., Dodson, R., Mika, A., Ord, S., Salvini, S., and Wieringa, M.: Calibration solvers and MS-MFS CLEAN in dataflow, Tech. Rep. SKA-TEL-SDP-00000??, SDP Consortium, draft dated 2016-12-??, 2016.

Offringa, A., McKinley, B., Hurley-Walker, N., Briggs, F., Wayth, R., Kaplan, D., Bell, M., Feng, L., Neben, A., Hughes, J., et al.: WSCLEAN: an implementation of a fast, generic wide-field imager for radio astronomy, Monthly Notices of the Royal Astronomical Society, 444, 606–619, 2014.

Ord, S.: ASKAP Approach to Gain Calibration, Tech. rep., CSIRO, https://confluence.ska-sdp.org/download/attachments/210698342/askap_gain_cal_v1.pdf, 2016.

Patil, A., Yatawatta, S., Koopmans, L., de Bruyn, A., Brentjens, M., Zaroubi, S., Asad, K., Hatef, M., Jelić, V., Mevius, M., et al.: Upper Limits on the 21 cm Epoch of Reionization Power Spectrum from One Night with LOFAR, The Astrophysical Journal, 838, 65, 2017.

Van Weeren, R., Williams, W., Hardcastle, M., Shimwell, T., Rafferty, D., Sabater, J., Heald, G., Sridhar, S., Dijkema, T., Brunetti, G., et al.: LOFAR facet calibration, The Astrophysical Journal Supplement Series, 223, 2, 2016.