MeerKAT Radio Telescope: First Science and Computing Overview

Simon Ratcliffe
Cutting Edge Optical Astronomy

MeerLICHT Optical Telescope, Sutherland
Everything we know so far

http://xkcd.com/273/
A fuller picture
Watt?

1 Jy = $10^{-26}$ Wm$^{-2}$Hz$^{-1}$
Here's one we made earlier...

Karoo Array Telescope
64 Dishes Complete – Inaugurated 13 July 2018
Square Kilometre Array
3 sites; 2 telescopes + HQ
1 Observatory

Design Phase: > €170M; 600 scientists+engineers

Phase 1
Construction: 2018 – 2024
Construction cost cap: €674.1M (inflation-adjusted)
Operations cost: under development (see below)

MeerKat integrated
Observatory Development Programme (€20M/year planned)
SKA Regional centres out of scope of centrally-funded SKAO.

Phase 2: start mid-2020s
~2000 dishes across 3500km of Southern Africa
Major expansion of SKA1-Low across Western Australia

Courtesy: Nick Rees, SKAO
HQ in UK; telescopes in AU & SA

- SKA1-LOW: 50 – 350 MHz
- Phase 1: ~130,000 antennas across 65km

SKA1-Mid: 350 MHz – 24 GHz
- Phase 1: 200 15-m dishes across 150 km

Construction: 2018 – 2024; Cost cap: €675M

Courtesy: Nick Rees, SKA0
An African proposition
Radio Telescopes – In a nutshell
\[ \mathbf{V}_{ij} = M_{ij} \mathbf{B}_{ij} \mathbf{G}_{ij} \mathbf{D}_{ij} \mathbf{E}_{ij} \mathbf{P}_{ij} \mathbf{T}_{ij} \mathbf{V}_{ij}^{\text{IDEAL}} \]
What is the Science Data Processor?
MeerKAT SDP Components

Simulators & Emulators
Execution Framework
Data Ingest
Calibration
Imaging
Time Domain
QA & Commissioning
Science Archive + Vis Storage
Hardware
8 Engineers
Calibrated, flagged, full time and spectral resolution visibility data (medium term storage) + 10x reduced product (indefinite)

Full res is 0.5 Hz / 32,768 channels + per vis flags and weights (32k channel in Q3)

MVF4 (MeerKAT Visibility Format v4) native with MSv2 export.

Calibration Tables including bandpass, gain, delay and cross-pol terms.
Future MeerKAT Data Products (Q4 2018)

Continuum image pipeline product baseline subtraction & self-cal solutions.

Spectral line image cubes (up to 100 hrs joint)

Export formats likely to include HDF5, CASA image and FITS format

Reduction for QA will be done as standard

Full res spectral cube is 24 TB

No plan to store residuals, PSFs etc...

Once stable for QA, effort switches to improving **science quality** under guidance from LSPs (2019).
Flexible Data Science

- **Digitiser**: 10 bit SPEAD data
- **F-Engine**: 8+8 bit SPEAD data
- **B-Engine**: 8+8 bit SPEAD data
- **X-Engine**: 32+32 bit integer data
- **Ingest**: 32+32 bit float data
- **Cal**: 32+32 bit float data

- **Channelised 32k / 4k**: 14 Gbps per pol, 1 x 40 GbE per engine
- **Visibility data @ 2 Hz**: Time x Freq x Baseline, 32 Gbps avg., 128 Gbps burst
- **Cal tables into tstate**: B,G,K etc., Self-cal from continuum imager

- **ADC Samples @ 1712 Gsps**: 17 Gbps per pol, 2 x 10 GbE per pol, Requires interleaving
- **Channelised 4k data**: 14 Gbps per pol, 1 x 40 GbE per engine, Time x Freq x Time
- **Visibility data @ 0.5 Hz**: Flags and precision, 11 Gbps avg., 22 Gbps burst

**40 GbE AoC from CBF Switch**

**40 GbE AoC from SDP Core Switch**
SDP Data Rates

- **Digitiser**
  - 64 antennas
  - 10 bit SPEAD data
  - 2.2 Tbps

- **Correlator**
  - 32+32 bit integer data
  - 200 Gbps

- **SDP Ingest**
  - 32+32 bit float data

- **Cal**
  - 32+32 bit float data

- **Spectral Imager**
  - 7k x 7k x 32k cubes

- **Visibility data @ 2 Hz**
  - Time x Freq x Baseline
  - 32k Freq Channels

- **Cal tables into tstate B,G,K etc.. Self-cal from continuum imager**

- **ADC Samples @ 1712 Msp**
  - 17 Gbps per pol

- **Visibility data @ 0.5 Hz**
  - + Flags and precision
  - 11 Gbps avg.
  - Typical 8 hour obs gives 20-40 TByte Vis Data

- **Typical obs gives 12 TByte Image Cube**
System master controller with Apache Mesos underneath

Manages 1.5 PFlop compute, 3 Tbps network, 45 PB storage

Custom metrics to allow fine grain control of resources such as GPUs

Key aspect of flexible, commensal task management

Both online and batch scheduling is handled through a single framework.
# Mesos

## Cluster:
- **(Unnamed)**
- **Server:** 10.98.2.1:5050
- **Version:** 1.1.0
- **Built:** 4 months ago by `ubuntu`
- **Started:** 2 weeks ago
- **Elected:** 2 weeks ago

## Agents
- **Activated:** 12
- **Deactivated:** 0

## Tasks
- **Staging:** 0
- **Starting:** 0
- **Running:** 8
- **Killing:** 0
- **Finished:** 312
- **Killed:** 115

## Active Tasks

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>State</th>
<th>Started</th>
<th>Host</th>
</tr>
</thead>
<tbody>
<tr>
<td>array_1_bc856M4k-00000001</td>
<td>array_1_bc856M4k.sdp.bf_ingest.1</td>
<td>RUNNING</td>
<td>27 minutes ago</td>
<td>bfi1.sdp.mkat.karoo.kat.ac.za</td>
</tr>
<tr>
<td>array_1_bc856M4k-00000002</td>
<td>array_1_bc856M4k.sdp.cam2telstate.1</td>
<td>RUNNING</td>
<td>27 minutes ago</td>
<td>cal6.sdp.mkat.karoo.kat.ac.za</td>
</tr>
<tr>
<td>array_1_bc856M4k-00000004</td>
<td>array_1_bc856M4k.sdp.filewriter.1</td>
<td>RUNNING</td>
<td>27 minutes ago</td>
<td>cal5.sdp.mkat.karoo.kat.ac.za</td>
</tr>
<tr>
<td>array_1_bc856M4k-00000003</td>
<td>array_1_bc856M4k.sdp.cal.1</td>
<td>RUNNING</td>
<td>27 minutes ago</td>
<td>cal5.sdp.mkat.karoo.kat.ac.za</td>
</tr>
<tr>
<td>array_1_bc856M4k-00000005</td>
<td>array_1_bc856M4k.sdp.ingest.1</td>
<td>RUNNING</td>
<td>27 minutes ago</td>
<td>ing1.sdp.mkat.karoo.kat.ac.za</td>
</tr>
<tr>
<td>array_1_bc856M4k-00000006</td>
<td>array_1_bc856M4k.sdp.timeplot.1</td>
<td>RUNNING</td>
<td>27 minutes ago</td>
<td>ing1.sdp.mkat.karoo.kat.ac.za</td>
</tr>
<tr>
<td>array_1_bc856M4k-00000000</td>
<td>array_1_bc856M4k.sdp.telstate</td>
<td>RUNNING</td>
<td>27 minutes ago</td>
<td>cal6.sdp.mkat.karoo.kat.ac.za</td>
</tr>
<tr>
<td>mesos-ui.3ef201b2-f21-11e6-a43e-22495207da16</td>
<td>mesos-ui</td>
<td>RUNNING</td>
<td>2 weeks ago</td>
<td>cal2.sdp.mkat.karoo.kat.ac.za</td>
</tr>
</tbody>
</table>
Subarrays and Commensal Observing

Subarray 1 - 2 dish engineering array

- Schedule Block
- Schedule Block
- Schedule Block
- Schedule Block
- Schedule Block
- Schedule Block
- Schedule Block

Antenna acceptance testing

Subarray 2 - 62 dish

- Schedule Block
- Schedule Block
- Schedule Block
- Schedule Block

32,768 channel spectral imaging

- Schedule Block
- Schedule Block
- Schedule Block

32,768 channel spectral imaging

Engineering BF Mode

SB Schedule Block SB

Pulsar Timing - 4 beam - individual steering

SB SB Schedule Block SB

SB Schedule Block SB Schedule Block

Thunderkat - Commensal image plane transient search
?data-product-configure array_1 bc856M4k
m000,m001,m008,m014,m015,m018,m031,m063 4096 0.25 2 {
"cam.http":
{"camdata": http://10.97.1.14/api/client/1"},

"cbf.antenna_channelised_voltage":
{"i0.antenna-channelised-voltage":
"spead://239.9.11.1+31:7148"},

"cbf.baseline_correlation_products":
{"i0.baseline-correlation-products":
"spead://239.9.12.1+31:7148"},

"cbf.tied_array_channelised_voltage":
{"i0.tied-array-channelised-voltage.0x":
"spead://239.9.13.1+31:7148",
"i0.tied-array-channelised-voltage.0y":
"spead://239.9.14.1+31:7148"}
}
Task Parallel

```
G = nx.DiGraph()

# Nodes: These represent tasks within the SDP
# that need to be configured. Most will be
# launched as containers on appropriate hosts.
G.add_node('sdp.telstate',
    db_key='db',
    docker_image='redis',
    docker_params={
        'port_bindings':
            [{
                'get_port':('redis',6379)}]},
    docker_host_class='sdpc')

# redis node for telescope state for graph
G.add_node('sdp.ingest.1',
    output_int_time=12,
    continuum_factor=32,
    docker_image='katcpd/ingest',
    docker_host_class='nvidia-dp',
    docker_params={
        'network':
            'host',
        'devices':
            ['/dev/nvidia0',
             '/dev/nvidia1',
             '/dev/nvidia2']})

# ingest node for ar1
# Edges: These describe the network layout
# between nodes, including import pairs and
# traffic types.
G.add_edge('sdp.ingest.1',
    'sdp.telstate',
    format={
        # get_multicast_ip('sdp.ingest.1'),
        # get_port('sdp.ingest.1')},
    input_rate=12800)

# spread data from xenom1 to ingest node
```

# Each node will have corresponding
# specific configuration. e.g.

'sdp.file_writer.1':
    'docker_host_class': 'tape_archive',
    'docker_image': 'katcpd/file_writer',
    '10_speed': '225.100.100.255:30997',
    'output_dir': '/var/data/',
    'telstate': '172.17.42.1:30998'

'sdp.ingest.1':
    'antenna_mask': 'm0062,m0063',
    'antennas': 2,
    'cam_speed': '225.100.100.254:30996',
    'cbf_speed': '225.100.100.1:30999',
    'continuum_factor': 32,
    'docker_host_class': 'nvidia_gpu',
    'docker_image': 'katcpd/ingest',
    'input_rate': 100000000, 0,
    '10_speed': '225.100.100.255:30997',
    'output_int_time': 2,
    'telstate': '172.17.42.1:30998'}
Telescope State Repository

Using redis as key→ts→value store

Implemented using zrange

Provides static configuration for stateless components

Dynamic data from upstream (CAM) and internal (e.g. CAL)

State repository per subarray product (graph)

Library provides namedpaced views for convenience
import katsdptelstate

ts = katsdptelstate.TelescopeState('localhost:30999')

[k for k in ts.keys() if k.startswith('cal_product')]

Out[3]: ['cal_product_B', 'cal_product_G', 'cal_product_K']

In [4]: ts['cal_product_B'].shape
Out[4]: (4096, 2, 8)

In [5]: ts.get_range('m025_pos_actual_scan_azim', st=1464102988, et=1464102990)

Out[5]: 
[(-70.4375975927352, 1464102988.100339),
 (-69.4169206963734, 1464102988.610161),
 (-68.4194805637115, 1464102989.110161),
 (-67.4006791367039, 1464102989.620024)]

In [6]: ts['subarray_product_id']
Out[6]: 'array_1_bc856M4k'

In [7]: ts['obs_params']
Out[7]: "sb_id_code '20160524-0030'"
Calibration Strategies

Calibration pipeline is essentially free-running, with strategy dictated by target.

Strategies include `bpcal`, `gaincal`, `polcal` and dedicated target tag for beamformer phase up – `bfcal`.

Solutions stored in `telstate` in appropriate namespace with time tagging.

Realtime usage (e.g. beamformer phase up) handled by `obs` script pulling from `telstate` and applying to CBF.
Key aspect of pipeline is 2D flagging.

1D flags from ingest are used to seed flagger.

Pipeline is as follows:

Flag Cal → Calc Cal Solutions → Apply Cal → Flag Target

Most memory intensive nodes (768 GB – 15 min buffer)
Calibration Solutions
Continuum and Spectral Imaging

Continuum imaging pipeline (Obit based) used primarily to deliver LSM and self cal solutions for use in the spectral imager.

Due to spectral, temporal and baseline averaging, continuum challenge substantially reduced.

Extensive modeling of spectral pipeline to understand memory, disk, and compute bounds.
Resultant design (with 40% contingency) has 1.5 PFLOPs, 4TB RAM, 1 PB scratch, 300 GBps disk IO.
Spectral Imager Installation - After

Deployed from bare metal with **MAAS** and **Ansible** in 1 day
MAAS / Ansible Proving Ground

Full online system redeployment from MAAS and Ansible – single day of scheduled downtime – including OS upgrade
Algorithms and Efficiency

2hr MeerKAT Simulated Data: 365 GGPA/s at 55% efficiency.
Data Storage and WAN Movement

CHPC Rosebank

CEPH Client
- Apache: Load Balancing and Authorisation

CEPH GW 1
- S3 Gateway
- librados

CEPH GW 3
- S3 Gateway
- librados

SDP Data Store (CHPC)
- 21.1 Raw PB in 55 Storage Pods
- CEPH Luminous 3:1 Replication

Y1:
- 8.5 PB Vis Data (11 Mo)
- 1.5 PB Image Data (12 Mo)

KAPB Karoo

Vis Buffer Node
- Object Migrator
- List Writer
- Flag Writer
- Vis Object Writer

SDP Vis Buffer (KAPB)
- 1 PB in 3 Pods
- Short term protection for link outages ~ 10 days
- CEPH 2:1

Imaging Node 1 of 30
- Image Object
- Image Object
- Image Object
- Image Object Writer (FIFO)
- Converged CEPH 1PB SSD Backed

W1 Ingest Node 1 of 4
- Spectral Vis + weights (Numpy array)
- SPEAD Streamer (Vis)

Cal Node 1 of 4
- Flags I + II
- SPEAD Streamer (Flags)

Telescope State Node
- Telescope State (Redis)
- Telescope State Serializer
## CEPH Hardware Platform

<table>
<thead>
<tr>
<th>Feature</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key Features</strong></td>
<td></td>
</tr>
<tr>
<td>25 GbE (SFP+)</td>
<td></td>
</tr>
<tr>
<td>x48 SAS 3 connectivity</td>
<td></td>
</tr>
<tr>
<td><strong>Processor</strong></td>
<td>Haswell-EP 4C Intel Xeon 4 core 3.7 GHz</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>64 GB per node</td>
</tr>
<tr>
<td><strong>Networking</strong></td>
<td>Mellanox Network Interface Card, 25GbE, Single-Port</td>
</tr>
<tr>
<td><strong>Drive Config</strong></td>
<td>48 x 8TB HDDs</td>
</tr>
<tr>
<td></td>
<td>2 x 512 GB nvme SSDs</td>
</tr>
<tr>
<td><strong>Form Factor</strong></td>
<td>4U with 1500W PSU</td>
</tr>
</tbody>
</table>
Tape Library Futures

$5 per slot / HH SAS LTO-7 Drives
Data Access Layer (KATDAL)

- **Local Node**
  - Objects in FS

- **Vis and Flags from S3 or Local Storage**

- **Storage Gateway**
  - Apache
  - CEPH Client Nodes 1/2/3
    - RadosGW

- **KATDAL**
  - Open against URL or local RDB file
  - Lazy load of Vis data objects as required
  - DASK arrays
  - Local FakeREDIS Instance

- **Telescope State Node**
  - Telescope State (Redis)
    - Namespace per Program Block
    - Vis / Flag / Weight Obj URLs
    - Telescope Metadata
    - SDP Ancillary Products

- **Solid lines indicate typical access pattern. Dashed are for more specialist use cases.**

- **MSv2**
External (to site) data access
Simulators & Emulators

- **KATTELMOD**
  - Defines telescope system
  - Mix of simulated, emulated and real components

- **CBF Emulator**
  - Docker container
  - Titan X GPU

- **SPEAD Visibility Data**
  - (64 antennas x 32k channels @ 32 Gbps)

- **SPEAD Voltage Data**
  - (Single beam, Dual Pol @ 34 Gbps)

- **Real SDP**
  - Capability setup and control as per std. observation

- **Telescope State**
  - Temporally accurate population of telescope state

- **Std Observation Script**

- **Simulated AP**
  - Accurate timing
  - (slew, track), error states and modes

- **Emulated Receiver**
  - Very simple
  - Stubbed digitiser

- **SDP Provided**
  - TM Provided
Logging, alerting and alarms – ELK stack

logspout → logstash → elasticsearch → kibana
Any task can be sent SIGUSR1 to toggle debug level
Each node runs a Prometheus endpoint, and master controller has an aggregate endpoint of internal sensors. Consul used for service discovery.

Central Prometheus instance collects and stores metrics and runs alerting rules (reporting via slack and katcp).

Grafana used for display and production of aggregate sensors.
System Monitoring - Grafana
Just getting started...