Programming in SeaScape Environment

April 7th, 2021

Ansys

Info For Attendees

- Please join audio via Audio Broadcast option
- Please use the Q&A window to clear queries and one of the panelists will answer it.
  - Direct questions to all of panelists
- This training is for 2 hours.
  - Will break into 2 sessions of 50 mins each, with 10 mins Q&A at the end of all two.
- The slides and recording will be available at Ansys website within a week
  - Registered participants will be receiving emails with the link
- For offline follow up of queries, please reach out to your local AE or email rahul.rajan@ansys.com

People on Panel

- **Host**
  - Rahul Rajan – Lead Product Specialist

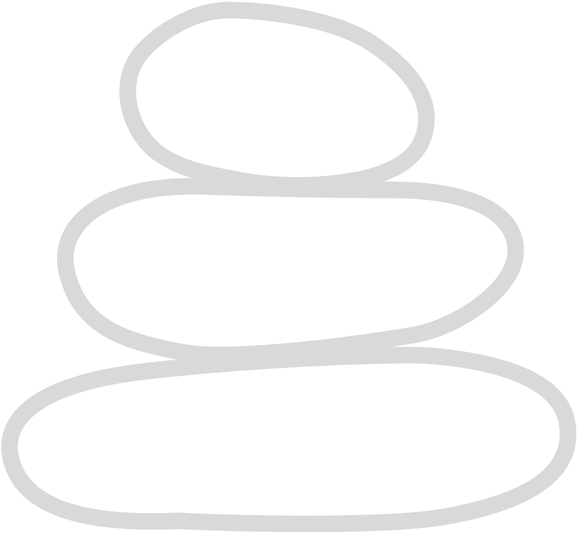
- **Panelists**
  - Anudeep Surasani – Lead Product Specialist
  - Sojan Philips – Lead Product Specialist

Table of Contents

- Classical EDA Tool Query System
- MapReduce
- “MapReduce Ready” SeaScape Data Structures
  - Heatmaps
  - ChunkedData
  - Creating Large Reports
- Writing Custom “Reduce” Functions
- Generic MapReduce Interface

Pre-Requisites for the Module

- RedHawk-SC Quick Start Training
- Basic Python – Session I
- Basic Python – Session II
- Design Modifications using RedHawk-SC



A Classical EDA Tool Query System

Serial Queries

- Find a set of instances that match certain properties

- Classical Way
  - Iterate over a collection of instances
  - Check the predicate for each instance
  - Build up a list with the instances that satisfies the predicate

Finding Clock Instances (the traditional methodology)

```python
import gp

# Find all the clock instances in the design
def find_clock_instances(instances, scn):
    items = list()
    for instance in instances:
        try:
            logic = scn.get_attributes(instance)
        except KeyError:
            continue
        if logic.get('clock_instance', False):
            items.append(instance)
    return items

instances = dv.get_instances()
clock_instances = find_clock_instances(instances, scn)
gp.gp_message('USER.000 Message', 1603795080, ('text', 'Number of Clock Instances: {0:1}'.format(len(clock_instances))))
```

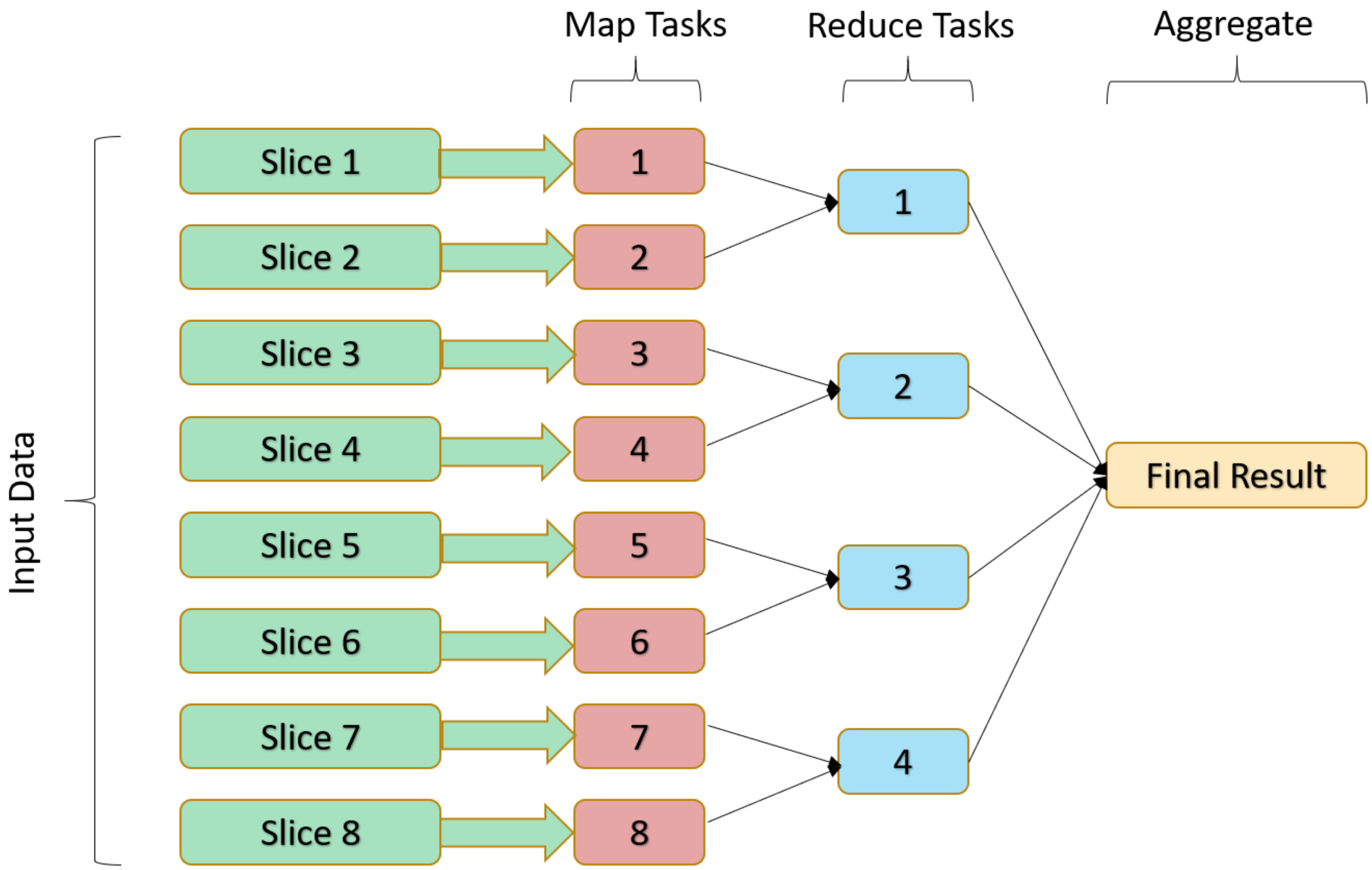
01_serial_queries.py

Can we do Better ?

- Instance Properties could be queried in parallel
- The jobs could be distributed over multiple machines

MapReduce

MapReduce Demystified



MapReduce in SeaScape

- What type of data am I going to Analyze ?
  - Set of Instances (Most Common)
  - Circuits (Actual Extracted Circuit)
  - Shapes (Geometries in the Design)
  - Custom/Generic (Generic, Sharded Data)

- What type of data am I going to return ?
  - Python list and dict are the usually the most common
  - Support for Stats, Waveforms and Others

- The size of the data being analyzed and returned have an impact on the walltime for a MapReduce Operation

Finding Clock Instances (using MapReduce)

```python
import gp

# Find all the clock instances in the design
def find_clock_instances(instances, scn):
    items = list()
    for instance in instances:
        try:
            logic = scn.get_attributes(instance)
        except KeyError:
            continue
        if logic.get('clock_instance', False):
            items.append(instance)
    return items

mm = gp.MapReduce(dv)
mm.map_reduce(dv.get_mr_instances(), partial(find_clock_instances, scn=scn))
clock_instances = mm.get()

gp.gp_message('USER.000 Message', 1603796237, ('text', 'Number of Clock Instances: {0:1}'.format(len(clock_instances))))
```

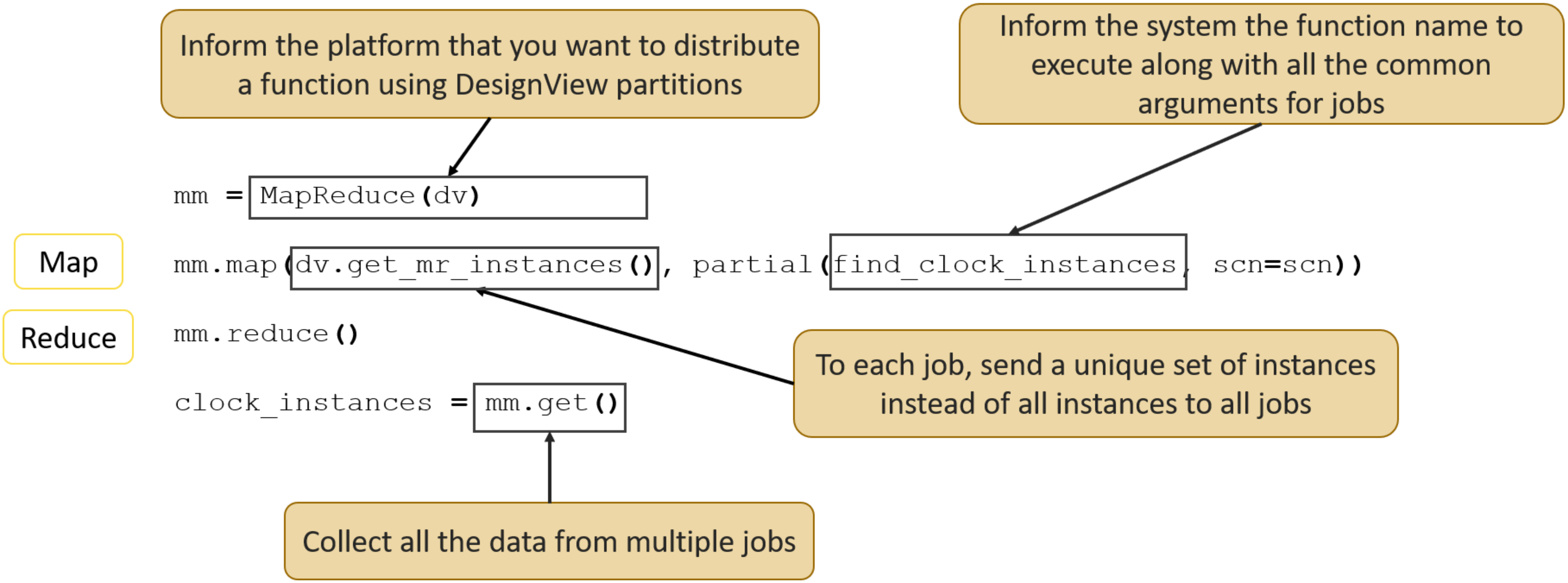
02_introduction_to_map_reduce_queries.py

Finding Clock Instances (Using MapReduce)

- The Python Partial Function
  - Convert a ‘n’ argument function to a ‘m’ argument function (m < n)

```
>>> from functools import partial
>>> def add(a, b): return a+b
>>> add(2, 3)
5
>>> add2 = partial(add, b=2)
>>> add2(6)
8
>>> add5 = partial(add, b=5)
>>> add5(2)
7
>>> add2(3) + add5(5)
15
```

Finding Clock Instances (Using MapReduce)



Inform the platform that you want to distribute a function using DesignView partitions

Inform the system the function name to execute along with all the common arguments for jobs

Map

Reduce

```
mm = MapReduce(dv)
mm.map(dv.get_mr_instances(), partial(find_clock_instances, scn=scn))
mm.reduce()
clock_instances = mm.get()
```

To each job, send a unique set of instances instead of all instances to all jobs

Collect all the data from multiple jobs

Finding Clock Instances (using MapReduce: A better way)

```
import gp

# Find all the clock instances in the design
def find_clock_instances(instances, scn):
    items = list()
    for instance in instances:
        if scn.get_attributes(instance)['logic'].get('clock_instance'):
            items.append(instance)
    return len(items)

mm = gp.MapReduce(dv)
mm.map_reduce(dv.get_mr_instances(), partial(find_clock_instances, scn=scn))
num_instances = mm.get()


gp.gp_message('USER.000 Message', 1603796237, ('text', 'Number of Clock Instances: {0:1}'.format(num_instances)))
```

02_introduction_to_map_reduce_queries.py

SeaScape MapReduce Collections

- Instances – `MRInstanceCollection – dv.get_mr_instances()`
- Shapes – `MRGeomCollection – dv.get_mr_shapes()`
- Circuits – `MRCircuitCollection – ev.get_mr_circuits()`

SeaScape: Accessing Objects by Name and ID

- Objects in SeaScape can be referenced by their Name or by their Id
  - `Net(‘VDD’) == Net(173)`
  - `Pin(‘A’) == Pin(8)`
  - `Instance('cts_inv_551661067’) == Instance(21467)`

- Within a MapReduce Job, the objects are accessed always by their ids
  - Faster to do logic operations on integers rather than strings

- Converting between domains
  - `DesignView.convert_to_id`
  - `DesignView.convert_to_name`
  - `DesignView.convert_to_user`

Working with Geometries in SeaScape

```python
import gp

def get_bumps_on_shapes(design_view, layer, net):
    layer_id = design_view.convert_to_id(layer).get_id()
    net_id = design_view.convert_to_id(net).get_id()
    mm = gp.MapReduce(design_view)
    mm.map_reduce(design_view.get_mr_shapes(), partial(map_get_bumps_on_shapes,
                                                       dv=design_view, layer_id=layer_id, net_id=net_id))
    return mm

top_layer = Layer('metal12')
net = Net('VDD')
bumps_ = get_bumps_on_shapes(dv, top_layer, net)
bumps = bumps_.get()
```

03_map_reduce_on_shapes.py

Working with Geometries in SeaScape

```
# Find the center for all shapes in the specified layer
def map_get_bumps_on_shapes(shapes, dv, layer_id, net_id):
    items = list()
    net_obj = gp.Net(net_id)
    for layer_geoms in shapes.get_layer_geoms():
        if layer_geoms.get_layer().get_id() != layer_id:
            continue
        for trap in layer_geoms:
            if trap.get_net_id() != net_id:
                continue
            trap_bbox = Units().microns(trap.get_bbox())
            center = trap_bbox.get_center()
            net_name = dv.convert_to_name(net_obj)
            bump_name = 'bump_{0}_{1}_{2}'.format(net_name, center.x_, center.y_)
            spice_port = 'spice_port_{0}'.format(net_name)
            items.append((bump_name, center, spice_port))
    return [dict(net=net_obj, layer=gp.Layer(layer_id), add_bumps=items)]
```

03_map_reduce_on_shapes.py

Working with Circuits in SeaScape

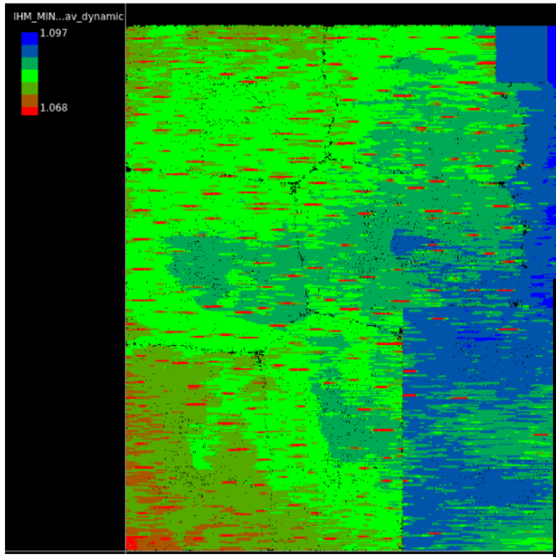
```python
import gp

# Scale the resistance of the specified layer by the specified multiplier
def map_scale_resistances(circuits, dv, r_factor):
    r_multiplier = { dv.convert_to_id(layer):factor for layer, factor in six.iteritems(r_factor) }
    if not r_multiplier:
        return
    for ckt in circuits:
        if ckt.is_coupling() or ckt.is_overlay():
            continue
        for edge in ckt.get_edges():
            factor = r_multiplier.get(edge.get_layer())
            if factor:
                edge.set_resistance(edge.get_resistance() * factor)

def scale_resistances(dv, r_factor=dict()):
    return partial(map_scale_resistances, dv=dv, r_factor=r_factor)

r_factor = { Layer('metal12'):0.8, Layer('metal1'):2.0 }
db_tmp = gp.open_db('tmpdb', enable_save=False)
mev = db_tmp.create_modified_extract_view(ev, eco_function=scale_resistances(dv, r_factor=r_factor), tag='mev')
```

04_map_reduce_on_circuits.py

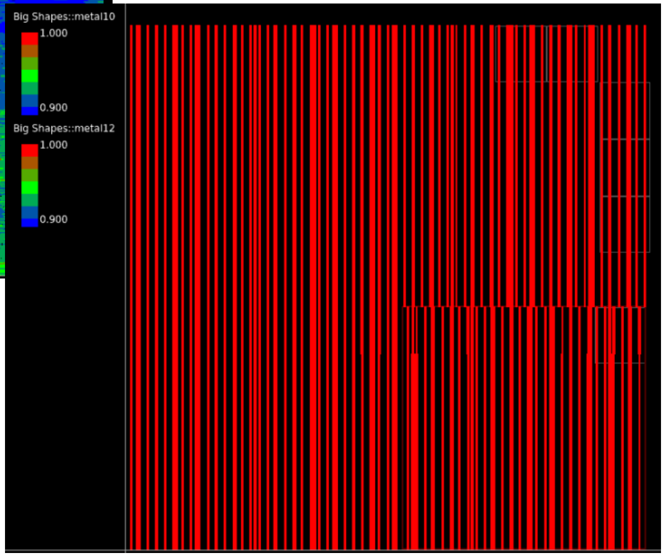
Built-in “MapReduce-Ready” Data Structures

Heatmaps

- Key-Value Data Structure
- Can be Visualized in the Layout GUI
- Creation and Querying is distributed
- Instance Heatmap
- Geometry Heatmap

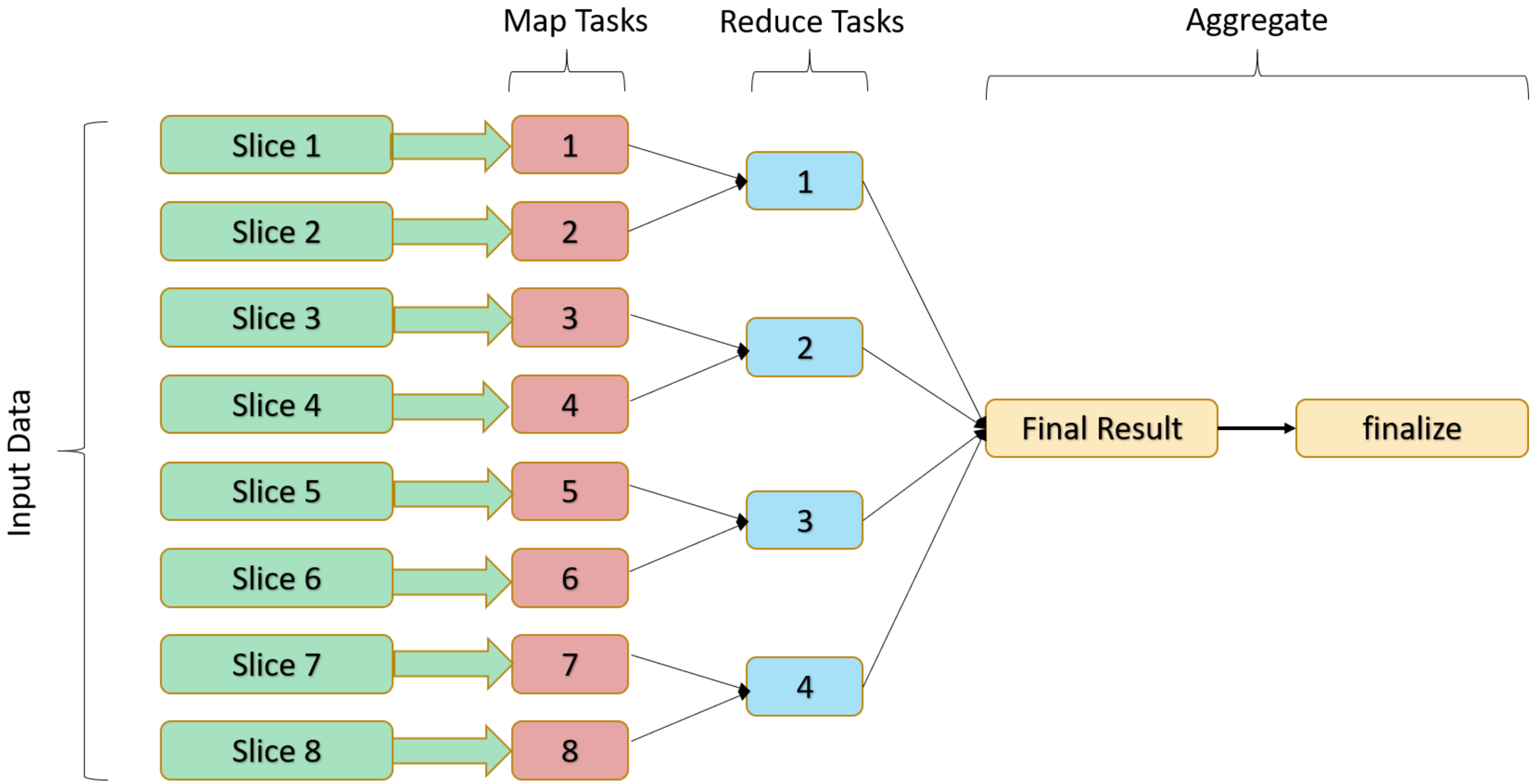


Instance Heatmap



Geometry Heatmap

MapReduce: The `finalize` function



MapReduce: The `finalize` function

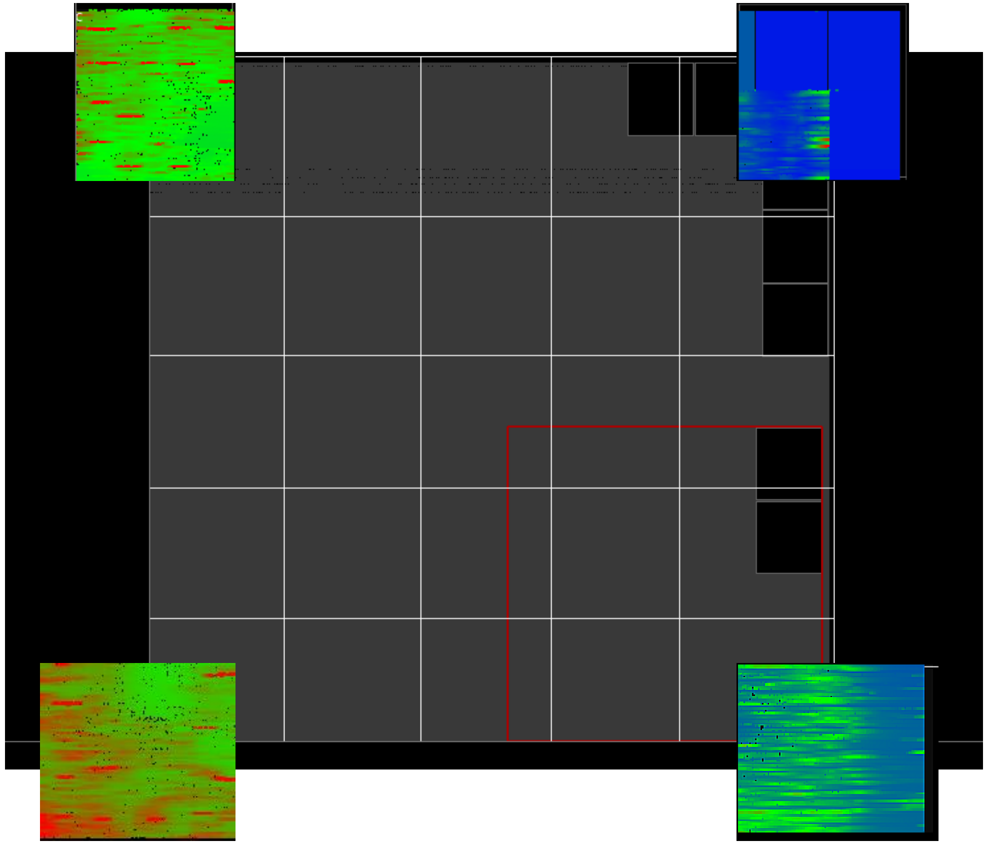
- Optionally called after aggregating the results from `reduce` step
- Helps the user apply a post-processing function to the result
  - e.g. Convert id-based to name-based
- SeaScape provides automatic finalize for commonly used data structs:
  - Heatmaps, `ChunkedData`, `gp_distributed_file`
- Only specify finalize_data if need to override default behavior:

```
mm.reduce(finalize_data=custom_finalize)
def custom_finalize(result):
    return process_result(result)
```

Heatmaps: Creation

map.part.0.4

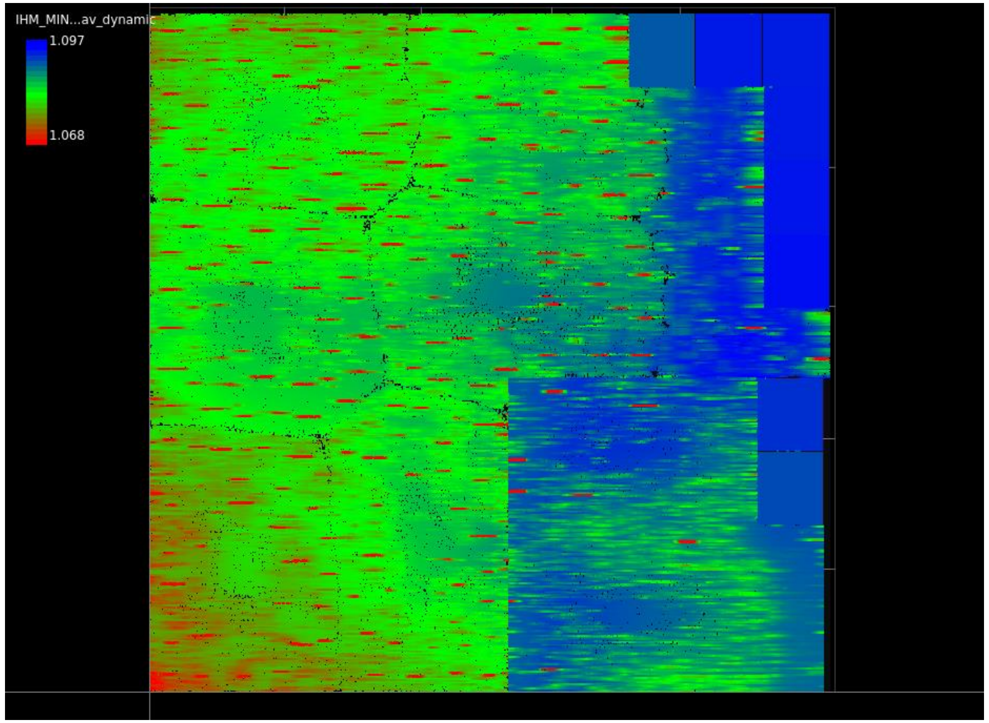
map.part.4.4



map.part.0.0

map.part.4.0

Reduce & Finalize



Creating an InstanceHeatmap

```
def create_part_inst_heatmap(instances, hm, dv):
    items = list()
    for instance in instances:
        value = dv.convert_to_id(instance).get_id()
        items.append((instance, value))
    hm.add_partial_data(instances, items=items)
    return hm

mm = gp.MapReduce(dv)
hm = gp.InstanceHeatmapPart(dv)
mm.map_reduce(dv.get_mr_instances(), partial(create_part_inst_heatmap, hm=hm, dv=dv))

heatamap = mm.get()
```

Creating an InstanceHeatmap

```python
def create_hybrid_voltage_heatmap(av):
    dv = av.get_related_views(gp.DesignView)[0]
    ihm = gp.InstanceHeatmapPart(dv)
    mm = gp.MapReduce(dv)
    mm.map(dv.get_mr_instances(), partial(map_create_hybrid_voltage_heatmap, av=av, ihm=ihm))
    mm.reduce(finalize_data=gp.mr_finalize_dict)
    return mm


def map_create_hybrid_voltage_heatmap(instances, av, ihm):
    dv = av.get_related_views(gp.DesignView)[0]
    items = list()
    for instance in instances:
        cell = dv.get_attributes(instance)['cell']
        pg_arcs = dv.get_cell_pg_arcs(cell)
        conns = dict(dv.get_instance_connections(instance, has_geoms=True, pg_net=True))
        for ppin, gpin in pg_arcs.items():
            voltage = _get_instance_voltage(instance, av, ppin, gpin)
            if voltage:
                pnet = conns.get(ppin, gp.Net(0))
                items.append((instance.get_id(), pnet.get_id(), voltage))
    ihm.add_partial_data(instances, items=items)
    return ihm
```

05_instance_heatmap.py

Creating a Geometry Heatmap

```python
def _map_create_geom_heatmap(shapes, hm, dv, pgnets, threshold):
    items = defaultdict(list)
    for layer_geom in shapes.get_layer_geoms():
        layer_id = layer_geom.get_layer().get_id()
        for trap in layer_geom:
            net_id = trap.get_net_id()
            if net_id not in pgnets or _get_bigger_size(trap.get_bbox()) < threshold:
                continue
            items[(layer_id, net_id)].append((trap, 1))
    hm.add_partial_data(shapes, items=items)
    return hm


def highlight_shapes(dv, threshold=100):
    pgnets = [dv.convert_to_id(xx).get_id() for xx in dv.get_nets('pg')]
    mm = gp.MapReduce(dv)
    hm = GeomHeatmapPart(dv)
    mm.map_reduce(dv.get_mr_shapes(), partial(_map_create_geom_heatmap, hm=hm, dv=dv, pgnets=pgnets,
threshold=threshold))
    return mm
```

06_geom_heatmap.py

Creating Large Reports

- When writing out large sets of data, reduction step can be the bottle-neck
  - Single point for data accumulation
  - Loop over the accumulated data and write to file

- Addressing the Bottleneck
  - Map Operations already work on partitions
  - Partition data can be written to individual files
  - Multiple files can be combined quickly with GNU Coreutils’ `cat` command

- `gp_distributed_file`

Creating a Report with gp_distributed_file

```python
def _map_create_instance_report(instances, av, fp):
    dv = av.get_related_views(gp.DesignView)[0]
    partition_id = dv.get_partition_id(instances)
    fp.initialize_part(partition_id)
    for instance in instances:
        if not instance.is_leaf():
            continue
        value = _get_instance_voltage(av, instance)
        fp.write("{0:50} {1:.4f}\n".format(dv.convert_to_name(instance), value))
    fp.close()
    return fp


def create_instance_voltage_report(av, file_name='./voltage.rpt'):
    dv = av.get_related_views(gp.DesignView)[0]
    fp = gp.gp_distributed_file(gp_util.fix_path(file_name))
    mm = gp.MapReduce(dv)
    mm.map_reduce(dv.get_mr_instances(), partial(_map_create_instance_report, av=av, fp=fp))

create_instance_voltage_report(av_dynamic)
```

07_gp_distributed_file.py

Common Reduction Functions

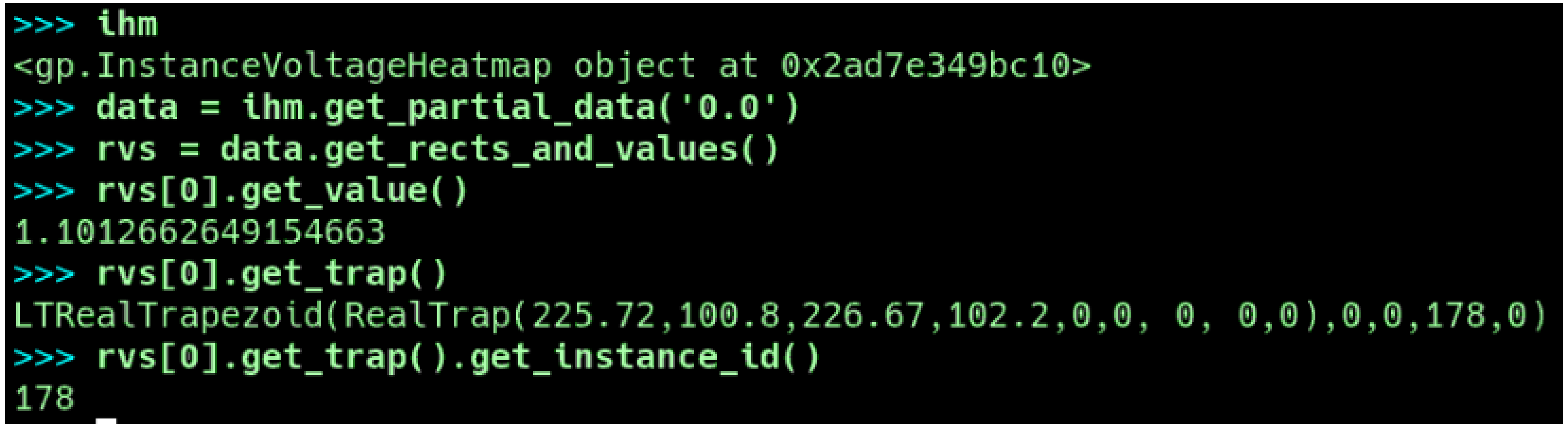
- Quick Review of MapReduce Paradigm
  - The input data is sharded/partitioned into smaller chunks
  - A logical 'map' operation is applied to each chunk
  - The result of the operations from each job is reduced and combined
  - An optional 'finalize' function is called on the combined result

- Reduction is the application of a reducing operation on the results of multiple Map Job Outputs

Common Reduction Functions

- reduce_add_dict (default for mm.reduce function)
  - Automatic travelsal of dict objects to add each key's value separately. It works for multi-level dicts also.
- reduce_max_dict
- reduce_sorted
- reduce_sorted_by_func

Retrieving Data from an InstanceHeatmap



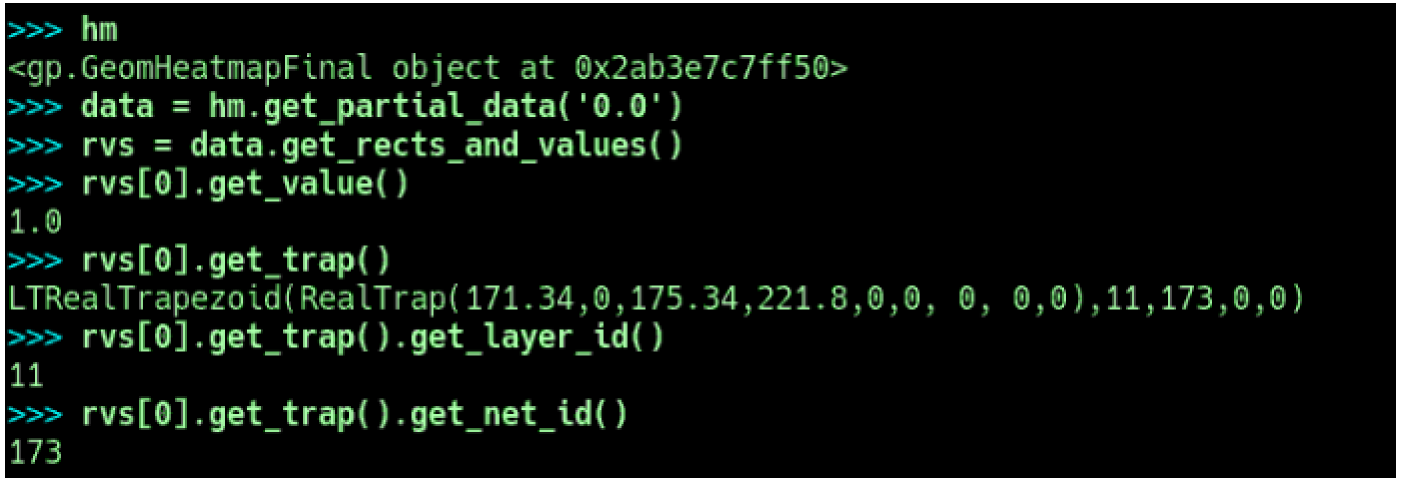
```
>>> ihm
<gp.InstanceVoltageHeatmap object at 0x2ad7e349bc10>
>>> data = ihm.get_partial_data('0.0')
>>> rvs = data.get_rects_and_values()
>>> rvs[0].get_value()
1.1012662649154663
>>> rvs[0].get_trap()
LTRealTrapezoid(RealTrap(225.72,100.8,226.67,102.2,0,0, 0, 0,0),0,0,178,0)
>>> rvs[0].get_trap().get_instance_id()
178
```

Retrieving Data from an InstanceHeatmap

```python
def create_instance_voltage_report(av, file_name='./voltage_stats.rpt'):
    ihm = av.get_instance_voltage_heatmap(data_type='eff_dvd')
    dv = av.get_related_views(gp.DesignView)[0]
    mm = gp.MapReduce(dv)
    fp = gp.gp_distributed_file(gp_util.fix_path(file_name))
    mm.map_reduce(dv.get_mr_instances(), partial(_map_create_instance_voltage_report, dv=dv, ihm=ihm,
fp=fp))


def _map_create_instance_voltage_report(instances, dv, ihm, fp):
    partition_id = dv.get_partition_id(instances)
    hm_data = ihm.get_partial_data(partition_id)
    rvs = hm_data.get_rects_and_values()
    items = list()
    fp.initialize_part(partition_id)
    for rv in rvs:
        voltage = rv.get_value()
        instance_id = rv.get_trap().get_instance_id()
        instance = gp.Instance(instance_id)
        fp.write('{0:50} {1:7.4f}\n'.format(dv.convert_to_name(instance), voltage))
    return fp
```

08_retrieving_data_from_instance_heatmap.py

Retrieving Data from an GeomHeatmap



```
>>> hm
<gp.GeomHeatmapFinal object at 0x2ab3e7c7ff50>
>>> data = hm.get_partial_data('0.0')
>>> rvs = data.get_rects_and_values()
>>> rvs[0].get_value()
1.0
>>> rvs[0].get_trap()
LTRealTrapezoid(RealTrap(171.34,0,175.34,221.8,0,0, 0, 0,0),11,173,0,0)
>>> rvs[0].get_trap().get_layer_id()
11
>>> rvs[0].get_trap().get_net_id()
173
```

ChunkedData

- Distributed Data Container that can hold more complex data
  - Heatmaps are limited to a single value per instance

- Cannot be visualized

- Easy to save and re-use within the database
  - E.g.: `emir_reports.get_instance_voltage_data`

Creating a ChunkedData Object

```python
def create_av_chunked_data(av):
    uv = ChunkedData()
    dv = av.get_related_views(gp.DesignView)[0]
    mm = gp.MapReduce(dv)
    mm.map_reduce(dv.get_mr_instances(), partial(_map_create_av_chunked_data, av=av, uv=uv))
    return mm.get()



def _map_create_av_chunked_data(instances, av, uv):
    dv = av.get_related_views(gp.DesignView)[0]
    partition_id = dv.get_partition_id(instances)
    items = dict()
    for instance in instances:
        value = _get_instance_voltage(av, instance)
        items[instance] = value
    uv.add_chunk_data(partition_id, data=items)
    return uv
```

09_chunked_data.py

Generic MapReduce Interface

- The MapReduce flow can be used to parallelize generic tasks
    - For e.g., sorting

- Achieved through the `map_part` interface

Generic MapReduce Interface

```python
import random

values = [[random.random() for _ in range(10)] for _ in range(10)]

def sort(vv):
        return sorted(vv)

mm = gp.MapReduce()
for ii in range(10):
        mm.map_part(partial(sort, vv=values[ii]))
mm.reduce(gp.reduce_add_merge_sorted)
gp.gp_print(mm.get())
```

Retrieving from a ChunkedData Object

```
def write_chunked_data_to_file(chunked_data, file_name='./chunked_data.rpt'):
    mm = gp.MapReduce()
    fp = gp.gp_distributed_file(gp_util.fix_path(file_name))
    for chunk_id in chunked_data.get_chunks():
        mm.map_part(partial(map_write_chunked_data_to_file, chunked_data=chunked_data,
chunk_id=chunk_id, fp=fp))
    mm.reduce()


def map_write_chunked_data_to_file(chunked_data, chunk_id, fp):
    fp.initialize_part(chunk_id)
    data = chunked_data.get_chunk_data(chunk_id)
    for instance, voltage in data.iteritems():
        fp.write('{0:50} {1:7.4f}\n'.format(instance, voltage))
    fp.close()
    return fp
```

09_retrieving_data_from_chunked_data.py

ChunkedParser

- Helper Class to parse ASCII/gzipped files in parallel
  - Gather data from large report files

- Define the parser by
  - Declaring the size of the chunks
  - Declaring the break sequence to be used

ChunkedParser Example

```
def count_words_with_pattern(file_name='/usr/share/dict/words', pattern='aaa'):
    pattern = re.compile(pattern, re.I)
    mm = gp.MapReduce()
    parser = gp.ChunkedParser(gp_util.fix_path(file_name), chunk_size_in_mb=1, break_sequence='\n')
    for chunk_id in range(parser.get_num_chunks()):
        mm.map_part(partial(parse_chunk, parser=parser, chunk_id=chunk_id, pattern=pattern))
    mm.reduce()
    return mm.get()


def parse_chunk(parser, chunk_id, pattern):
    parser.jump_to_chunk(chunk_id)
    count = 0
    for line in parser:
        if re.search(pattern, line):
            count += 1
    return count
```

11_chunked_parser.py

Writing “fall-through” scripts

- `gp_delayed_object`
  - o SeaScape Specific object to Schedule Jobs/Functions on the Workers

- Allows fall-through (job execution at workers)
  - - Does not hold up the Master Console when executing

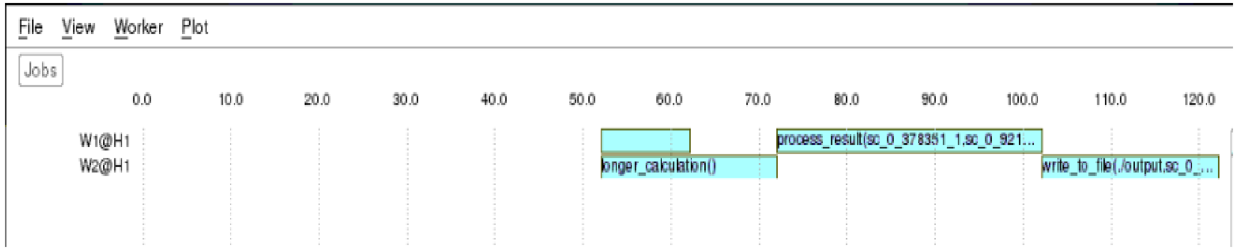
- Call `get` on the `gp_delayed_object` to use the results
- Gp_delayed_objects do automatic dependency detection for MapReduce or other GpDelayedObject types.
- Do not use gp_delayed_object directly.
- Prefer to use @sch_func decorator, which implicitly appends a gp_delayed_object when the function with @sch_func is called.

Serial Execution of Jobs

```python
def long_calculation(args):
    return do_something(args)

def longer_calculation(args):
    return do_something_else(args)

def process_result(aa, bb):
    return calculate(aa, bb)

def write_to_file(file_name, value):
    with open(file_name, 'w') as fp:
        fp.write(value + '\n')


intermediate_result_1 = long_calculation(args)
intermediate_result_2 = longer_calculation(args_2)
final_result = process_result(intermediate_result_1, intermediate_result_2)
write_to_file('output', final_result)
print("Calculated")
```

Making the Calculations fall-through with `gp_delayed_object`

```python
@sch_func
def long_calculation(args):
    return do_something(args)

@sch_func
def longer_calculation(args):
    return do_something_else(args)

@sch_func
def process_result(aa, bb):
    data_1 = aa.get()
    data_2 = bb.get()
    return calculate(data_1, data_2)

@sch_func
def write_to_file(file_name, value):
    with open(file_name, 'w') as fp:
        fp.write('{0}\n'.format(value.get()))

intermediate_result_1 = long_calculation(args)
intermediate_result_2 = longer_calculation(args_2)
final_result = process_result(intermediate_result_1, intermediate_result_2)
write_to_file('output', final_result)
print("Submitted")
```



Making the Calculations fall-through with `gp_delayed_object`

A More Practical Example

```
@gp.sch_func
def gather_instance_attributes(view, instances):
        result = dict()
        for instance in instances.get():
                result[instance] = view.get_attributes(instance)
        return result

@gp.sch_func
def write_report(file_name, freq, props):
        with open(file_name, 'w') as fp:
                fp.write('Dominant Frequency: {0.6e}'.format(freq.get())
                for kk, vv in props.iteritems():
                        fp.write('{0} {1}\n'.format(kk, vv))

dominant_freq = find_dominant_clock_frequency(scn)
topN = find_top_power_instances(scn)
props = gather_instance_attributes(scn, topN)
write_report("./my_report", dominant_freq, props)
```

Making the Calculations fall-through with `gp_delayed_object`

A More Practical Example

```
def find_dominant_clock_frequency(scn):
        dv = scn.get_related_views(gp.DesignView)[0]
        mm = gp.MapReduce(dv)
        mm.map_reduce(dv.get_mr_instances(), partial(m_find_dominant_clock, scn=scn)
        return mm

def find_top_power_instances(scn):
        dv = scn.get_related_views(gp.DesignView)[0]
        mm = gp.MapReduce(dv)
        mm.map(dv.get_mr_instances(), partial(m_find_top_power, scn=scn)
        mm.reduce(partial(gp.reduce_sorted, topN=100))
        return mm


dominant_freq = find_dominant_clock_frequency(scn)
topN = find_top_power_instances(scn)
props = gather_instance_attributes(scn, topN)
write_report("./my_report", dominant_freq, props)
```

Thank You !

- RedHawk-SC is an incredibly powerful analysis tool that you can wield to your heart's content
- The key to unlocking this superpower is to learn about the distributed data structures in RedHawk-SC and how to unlock them
- Anything that can be queried can be visualized or written out