Evaluating the suitability of MapReduce for surface temperature analysis codes

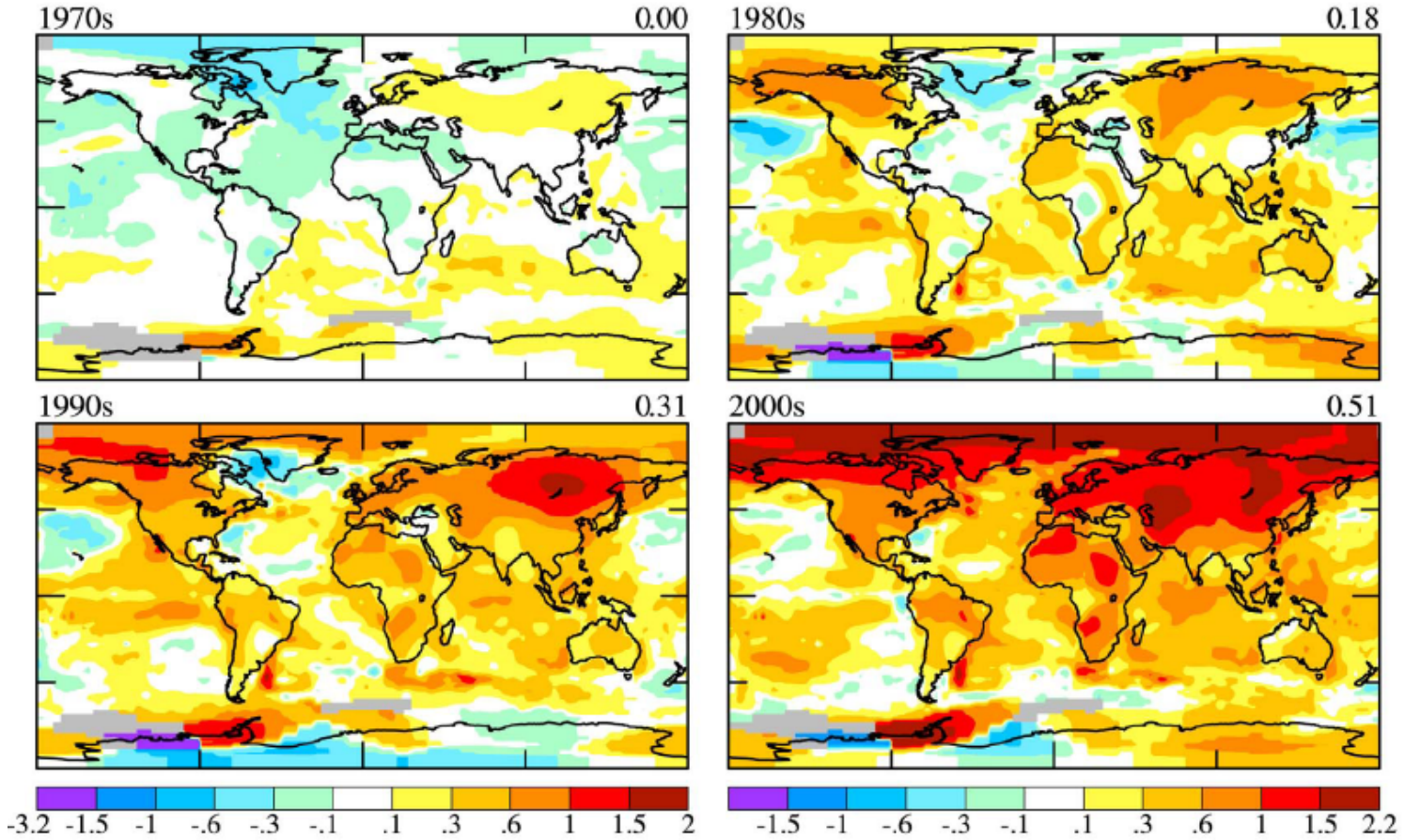
Vinay Sudhakaran and Neil Chue Hong

The Second International Workshop on
Data-Intensive Computing in the Clouds
14th November 2011, Seattle, USA

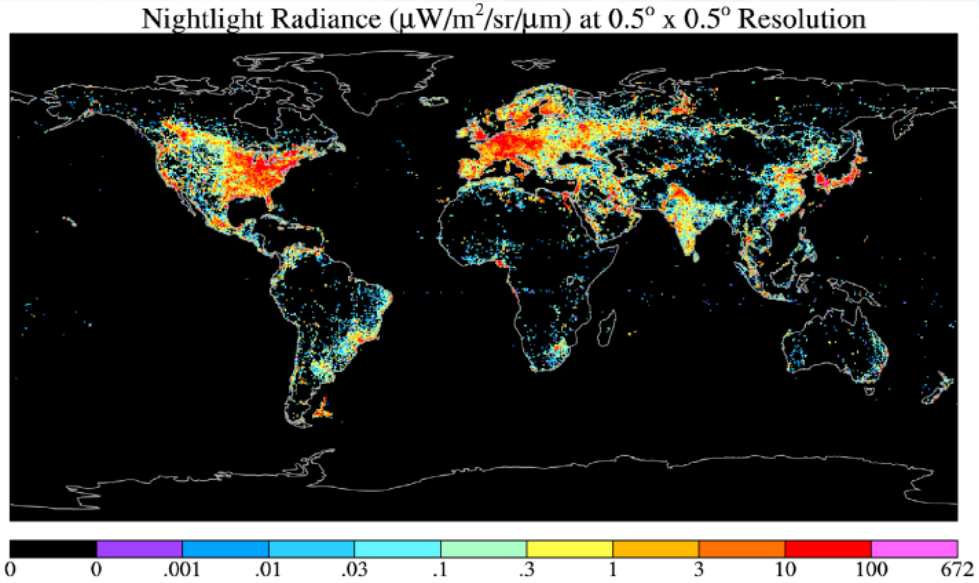
Neil Chue Hong
Director, Software Sustainability Institute
N.ChueHong@epcc.ed.ac.uk
+44 131 650 5957

Surface Temperature Analysis

- Global surface temperature anomalies relative to base
  - On average, successive decades 0.17 degrees warming
- Warming in recent decades is larger over land than ocean
  - US: 50%
  - Eurasia: 2x
  - Poles: 3x



Motivation for investigating MapReduce



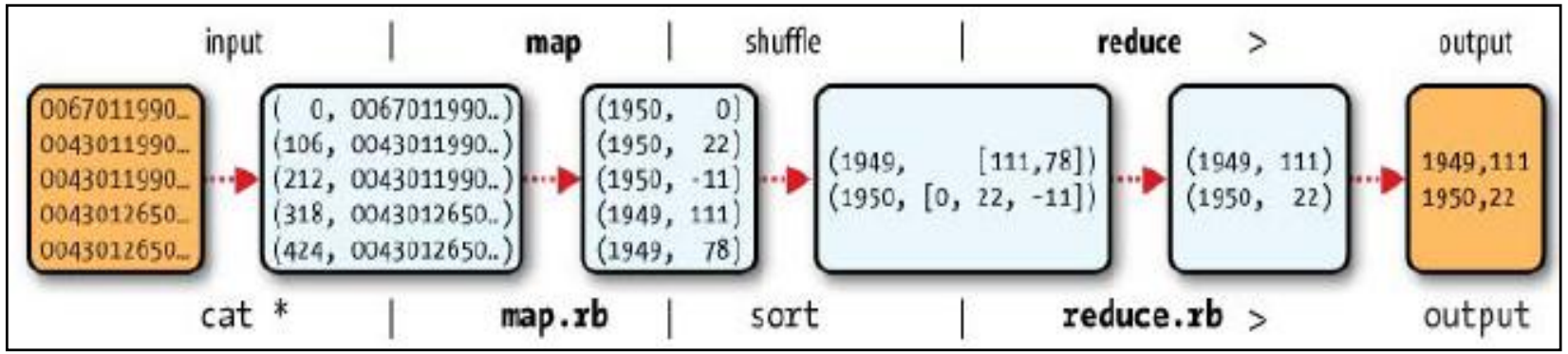
- Urbanisation significantly impacts accuracy of temperature measured by stations located in or near urban areas
- Global satellite measurement of night lights allow check of magnitude of urban influence
  - Perform adjustments on temperature data for “bright” urban stations to agree with the temperature data of nearby rural stations.
- This adjustment is relatively computationally intensive
  - But also has non-trivial (if simple) data access patterns
  - Interesting for D3Science/3DPAS and UK CloudSIG communities

MapReduce – Weather Data Example

- ***Map*** processes a key/value pair to generate a set of intermediate key/value pairs
- ***Reduce*** merges intermediate values associated with the same key
- Temperature record sets

```
(0, 0067011990999991950051507004...9999999N9+00001+99999999999...)
(106, 0043011990999991950051512004...9999999N9+00221+99999999999...)
(212, 0043011990999991950051518004...9999999N9-00111+99999999999...)
(318, 0043012650999991949032412004...0500001N9+01111+99999999999...)
(424, 0043012650999991949032418004...0500001N9+00781+99999999999...)
```

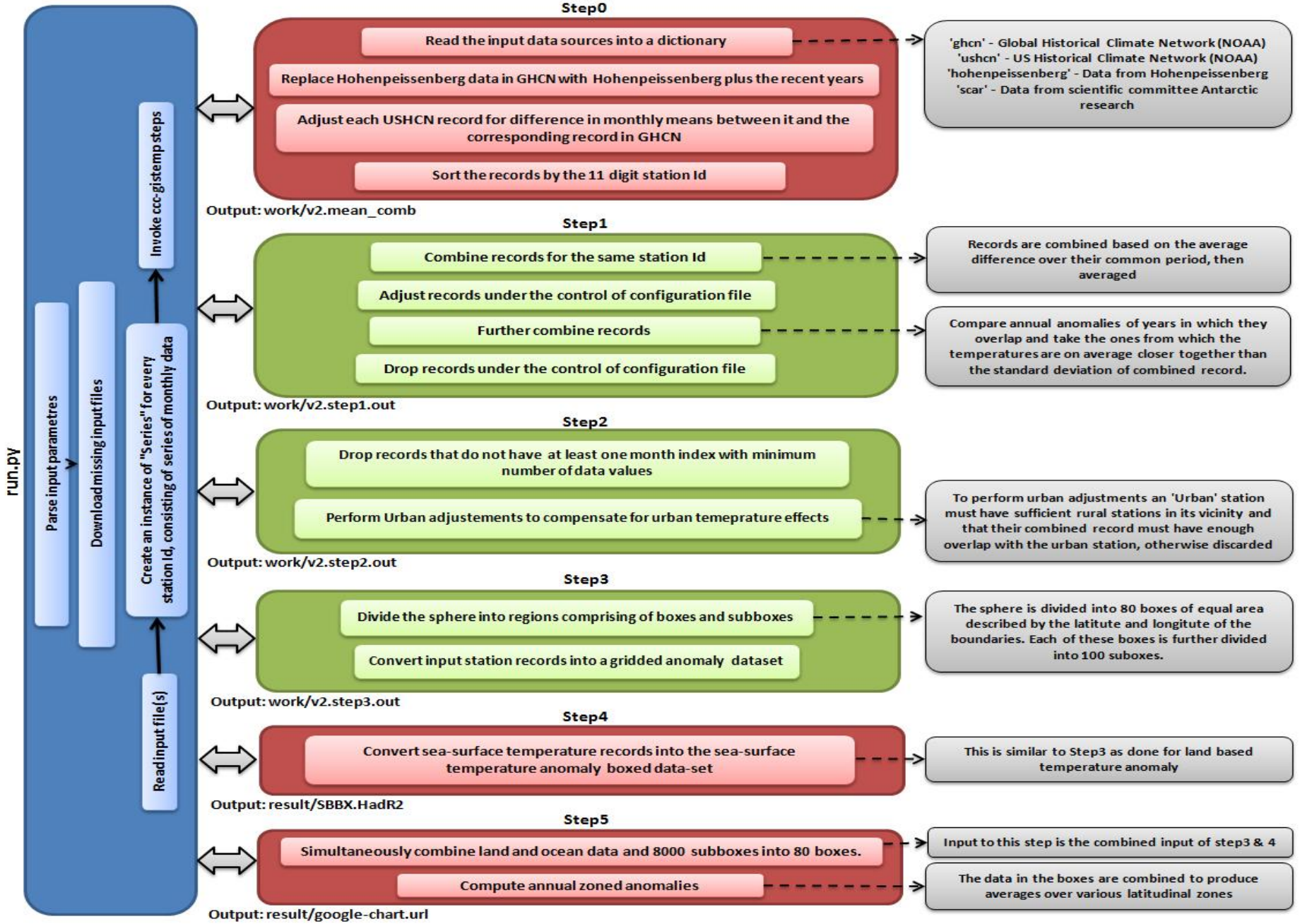
- Find highest temperature for each year



GISSTEMP/ccc-gistemp/mrjob

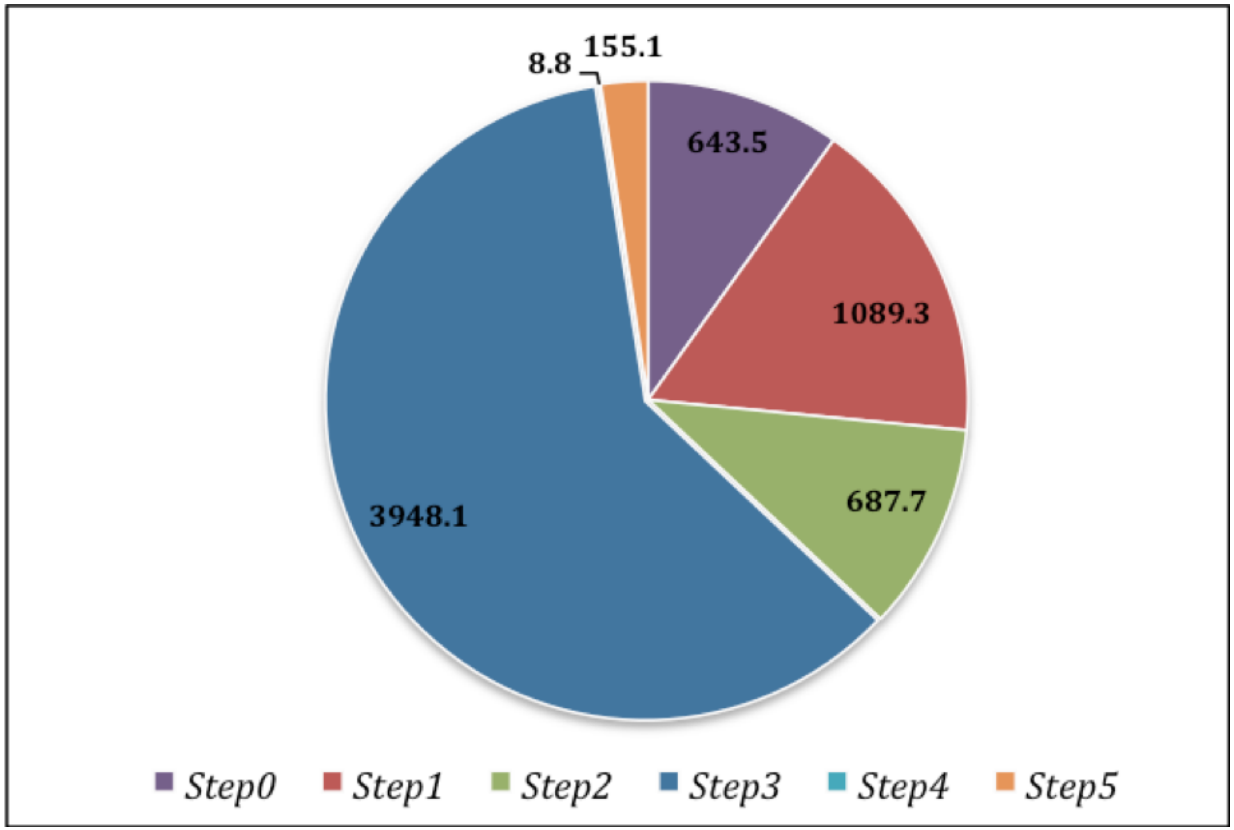
- Open-source model for estimating the global temperature change written in Fortran
  - Implemented by NASA Goddard Institute of Space Studies (GISS)
- ccc-gistemp is a re-implementation of the NASA GISTEMP in Python from Clear Climate Code (CCC) for improved clarity.
- Merits of ccc-gistemp:
  - Improved readability and maintainability
  - Efficient use of Python data-structures and iterators
  - Extensible programming style
  - Discussion groups to clarify doubts and share ideas
  - Almost identical results produced to that of the original GISS code
- *mrjob* is a package that provides a simple abstraction for writing MapReduce jobs in Python and Hadoop by defining steps for specifying 'mapper' and 'reducer' functions, input and output file format (protocol) and paths
  - ability to write multi-step jobs (one map-reduce step feeds into the next)
  - custom switches which can be added to jobs, including file options
  - Implements efficient cPickle module for serialising/deserialising complex data structures

ccc-gistemp workflow

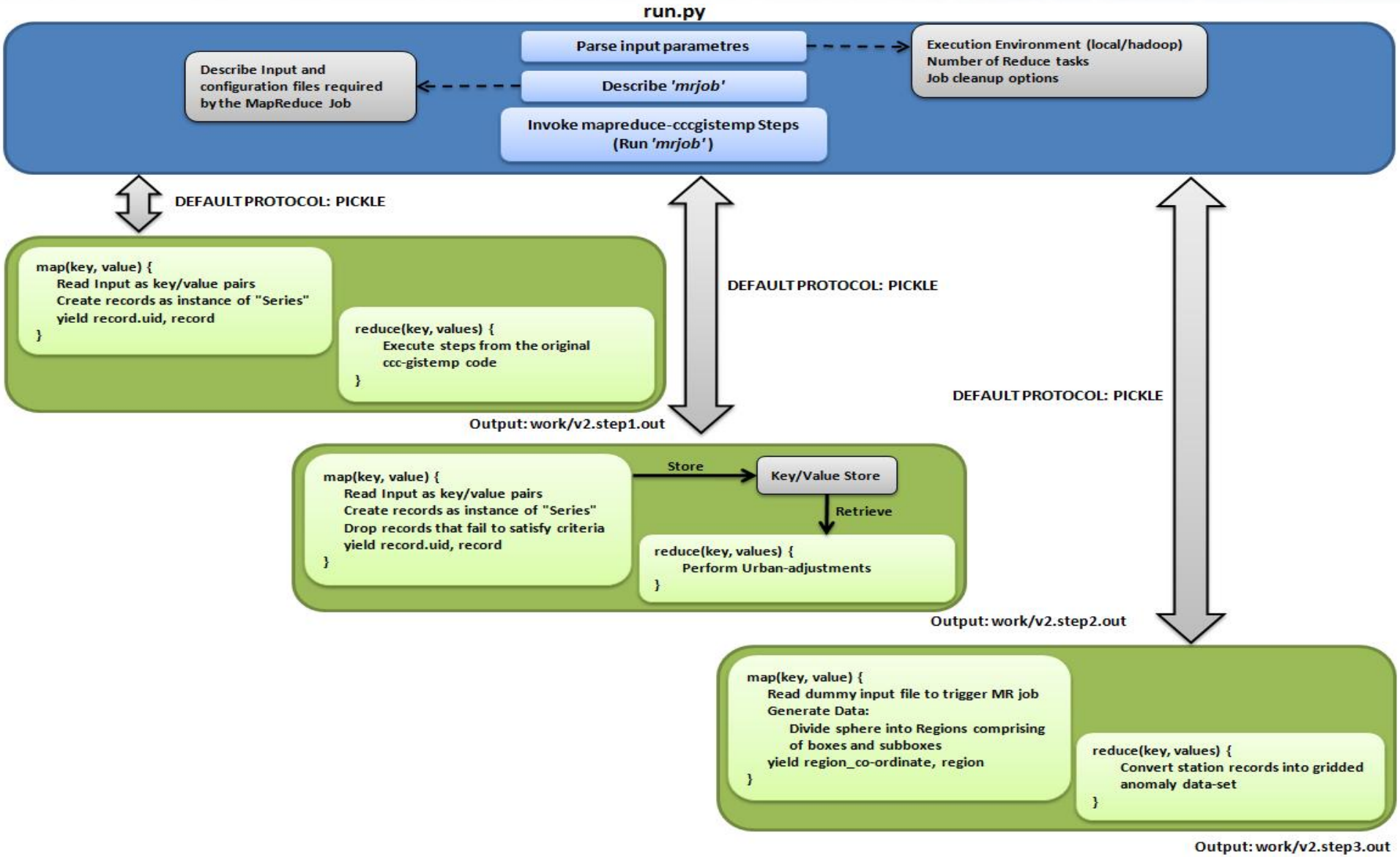


Profiling original code

- Majority of code spent in Steps 0, 1, 2 and 3
  - Step 0 is the pre-processing step where all datasets (GHCN, USHCN, SCAR, Hohenpeissenberg) are read and joined
  - Appears ideal for parallelisation but requires global synchronisation with a compare-merge operation
  - mrjob+hadoop not designed to simultaneously operate on two independent input sources
  - Step 0 not ported to MapReduce



Revised ccc-gistemp workflow



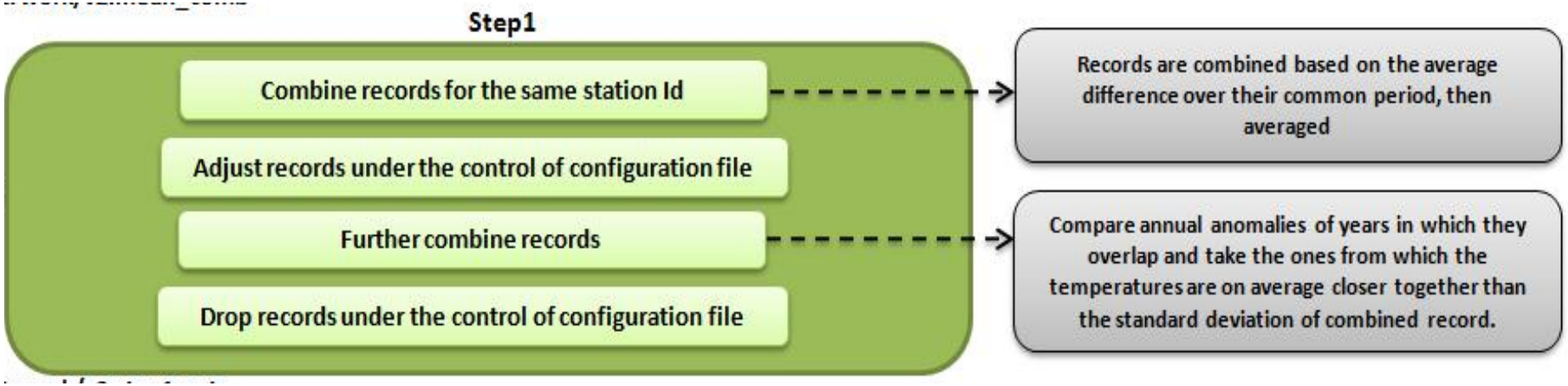
Benchmarking Infrastructure

- EDIM1: "Amdahl-balanced" cluster
- Used a 16 node partition of EDIM1
  - 1 master node, 1 job tracker, 14 slaves (28 cores)
  - Averaged over consecutive executions
  - Observed variance between runs always less than +/-1%
- Dataset: http://www.ncdc.noaa.gov/ghcnm/
  - 6000 temperature, 7500 precipitation, 2000 pressure stations
  - Many years of data (earliest 1697, 1650+ records greater than 100 years)
  - ~60MB in total ☺

Category	Configuration
Number of Nodes	120 (3 racks of 40 nodes each)
Processors/Node	Dual-Core Intel 1.6 GHz ATOM[42] processor
Disk Storage/Node	1 x 256 MB Solid State Drive (SSD)
	3 x 2TB HDD
Network	10 Gigabit Ethernet
OS	Rocks (Clustered Linux Distribution based on CENTOS)[43] Linux Kernel Version 2.6.37
JVM	1.6.0_16
Hadoop	0.20.2 Cloudera Distribution version 3 (CDH3)

Table 1: Hardware configuration of EDIM1 machine

Step 1: Combine station records

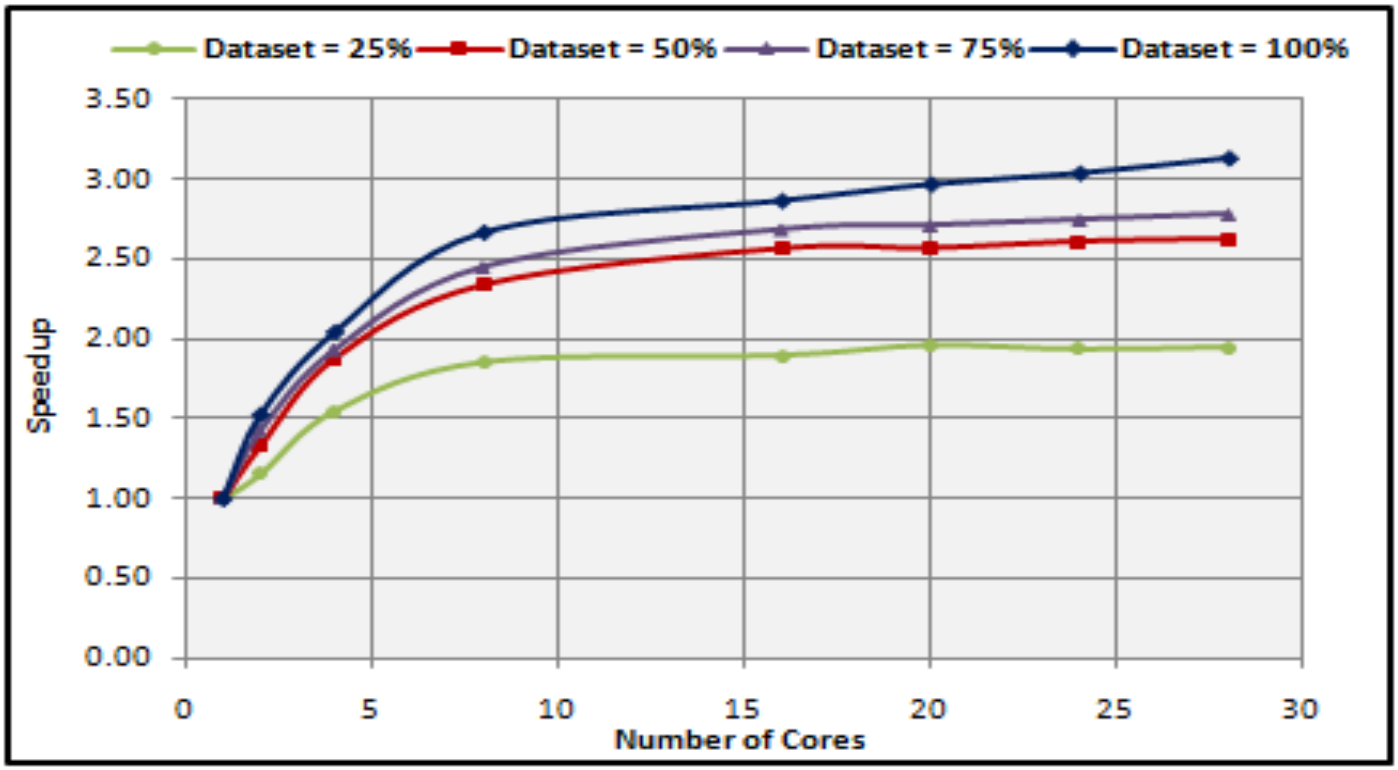


An instance of "Series" (Object)

12 Digit Unique Id (UID) 11 digit station Id + 1 digit Series Identifier	Year	Month											
		0	1	2	3	4	5	6	7	8	9	10	11	12
101603550002	1880	101	128	141-9999	160	204	220	228	210	199	154	119
	1881						:						
	:						:						
	2010	133	143	149	167	185	217	254	199	235	205	167-9999	
	2011	131	133	140	175	198	225	257	267	240	211	161-9999	

Step 1: Combine station records

- Trivially data-parallel and well suited to MapReduce.
  - Hadoop implementation incurs considerable start-up costs which are usually amortised when processing large amounts of data
  - However, if the data-set is small, these initial start-up costs dominate even when executed on large number of nodes (e.g. HDFS distribute)



Step 2: Clean data / urban adjustments

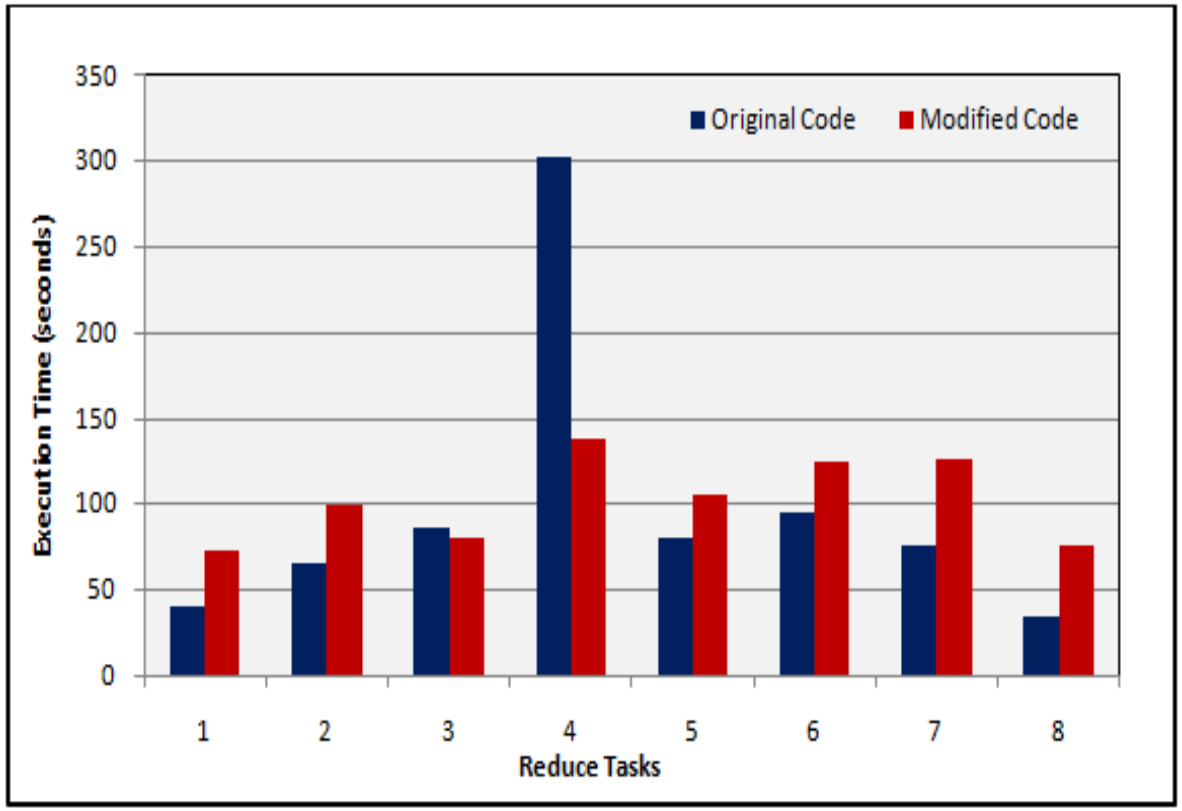
Step2

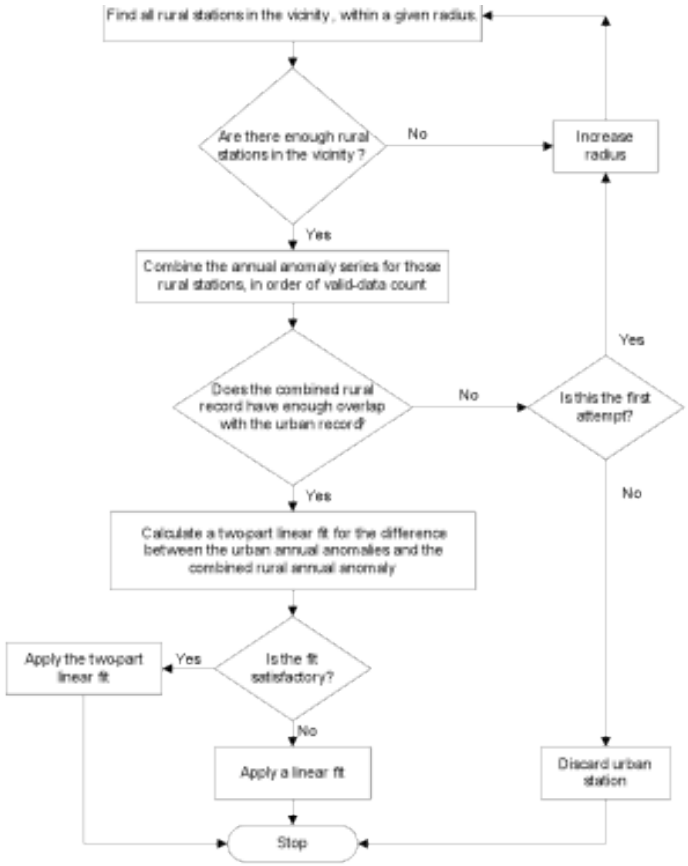
Drop records that do not have at least one month index with minimum number of data values

Perform Urban adjustements to compensate for urban temeprature effects

t: work/v2.step2.out

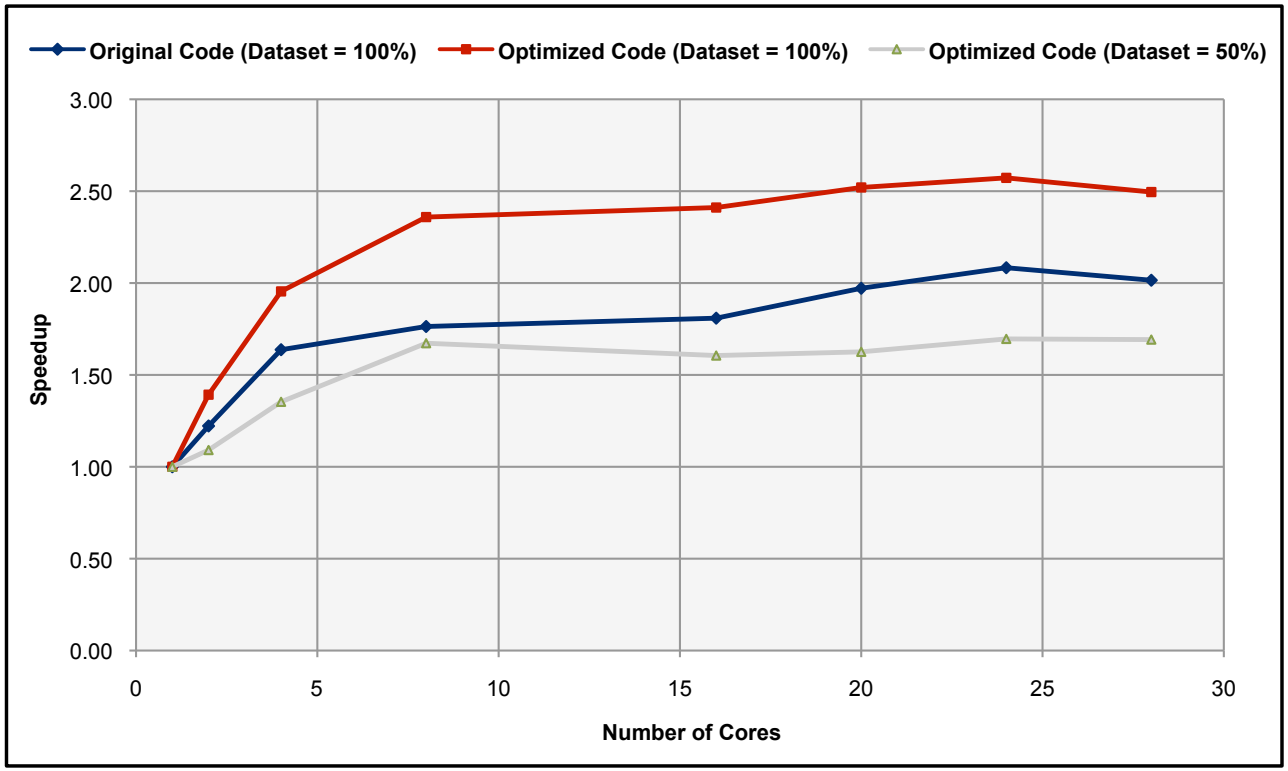
To perform urban adjustments an 'Urban' station must have sufficient rural stations in its vicinity and that their combined record must have enough overlap with the urban station, otherwise discarded



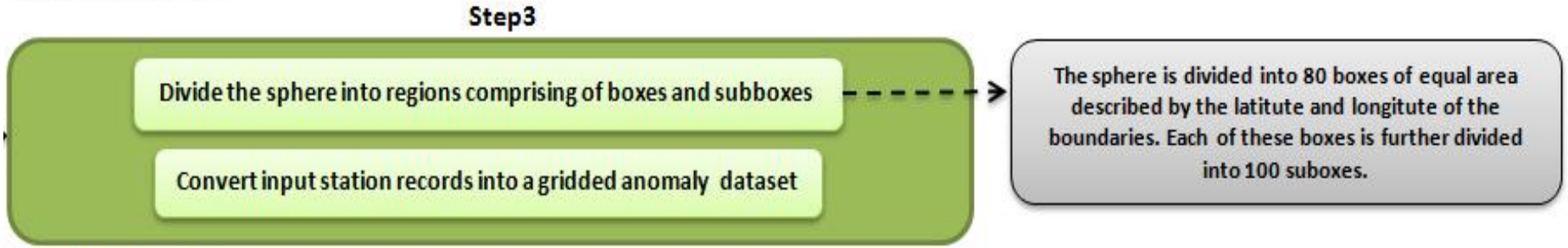


Step 2: Clean data / urban adjustments

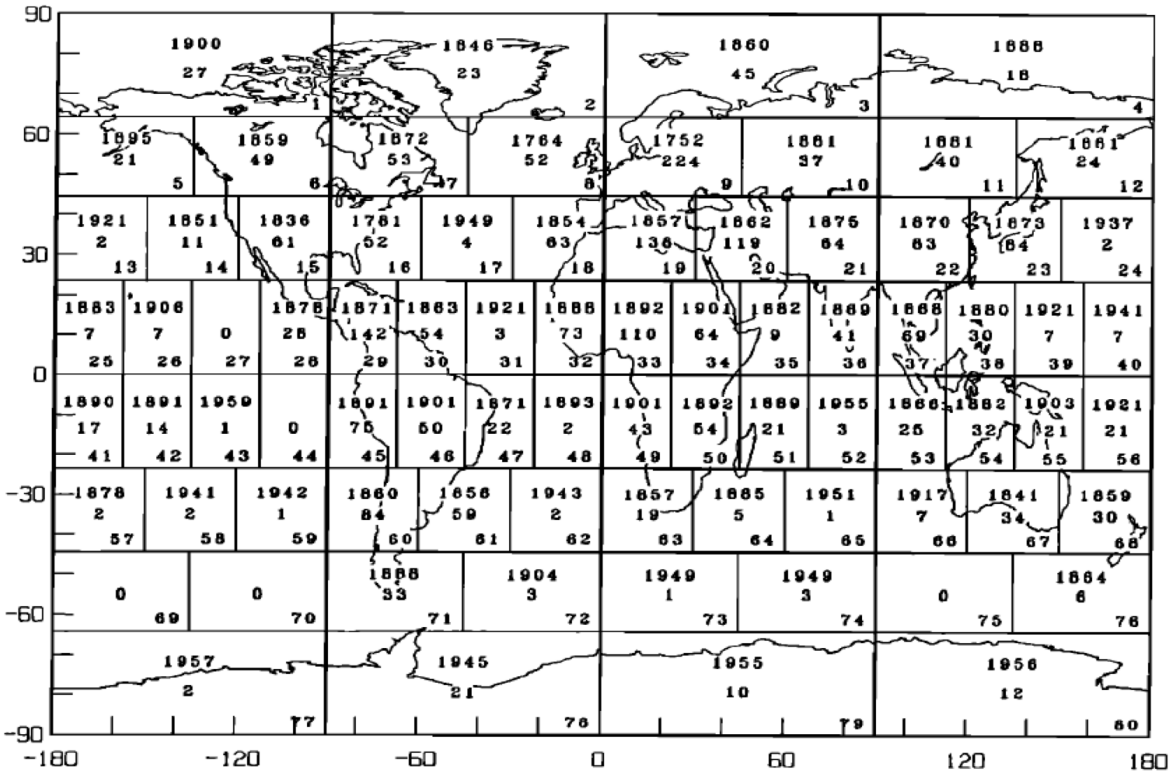
- Uneven distribution of workload due to uneven distribution of values associated with a ‘key’
  - MapReduce assigns all values associated with the same key to a single reduce task
  - Large number of records with station id beginning with ‘42’ (USA)
  - Code modified to use different part of station id as key
  - Merge Step 1 and Step 2 to avoid excessive “pickling/depickling”



Step 3: Convert to gridded anomaly sets



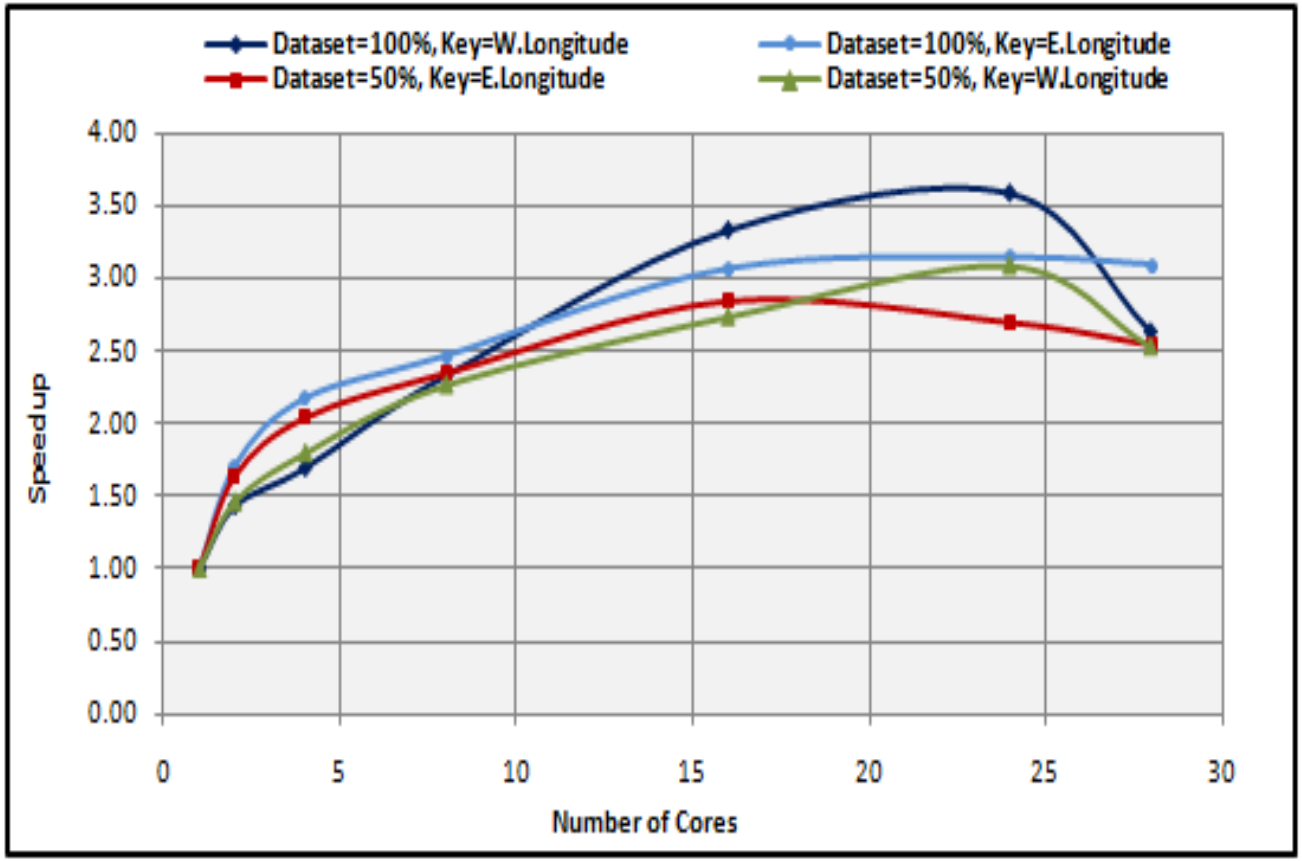
t: work/v2.step3.out



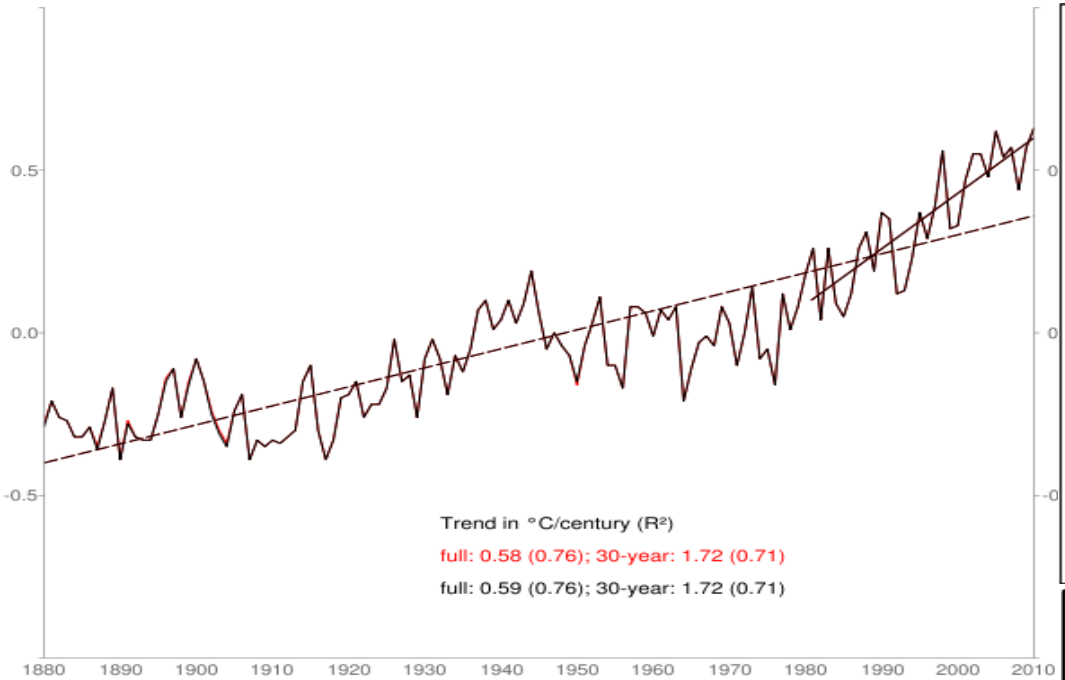
- Conventional approach
  - Output of Step 2 as input of Step 3
- Optimised approach
  - No direct input to Step 3
  - Read regions directly
  - Convert to key/value pairs
  - Choose 1 of the 4-tuples (lat/long) as key and tuple of region + associated subboxes as value
  - Choose longitude over latitude for better balancing

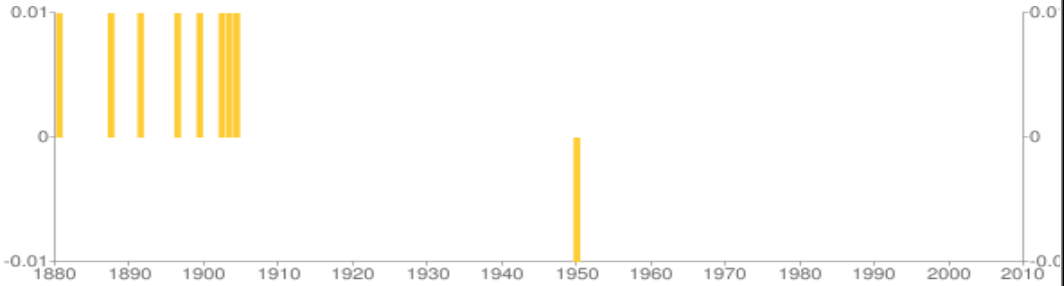
Step 3: Convert to gridded anomaly sets

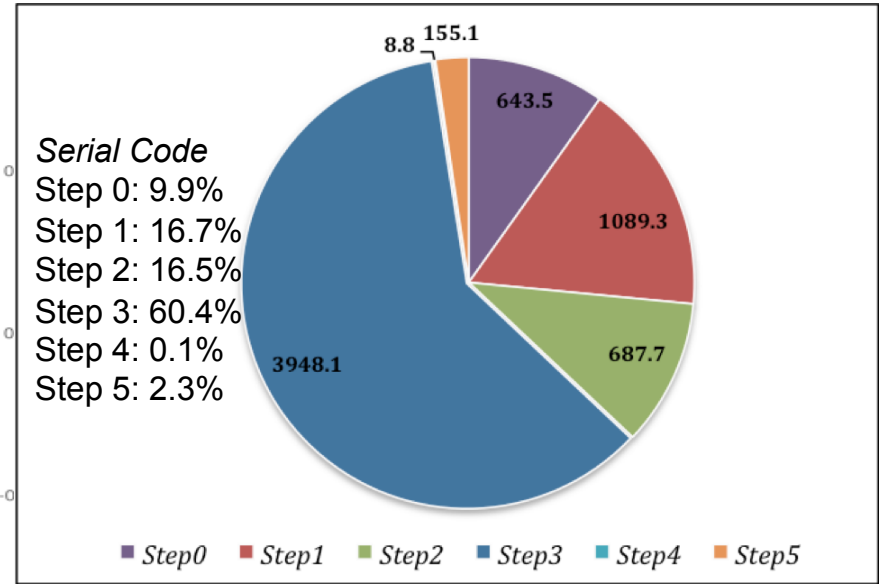
- Hadoop load balancing ideally suited for jobs that are large, but can be divided into smaller units of nearly equal size
  - A single large task can slow the overall performance
- Unique keys limited by longitudes dividing sphere
  - Scaling beyond maximum number of reduce tasks that can be created causes a significant decline in performance due to the presence of idle processing units

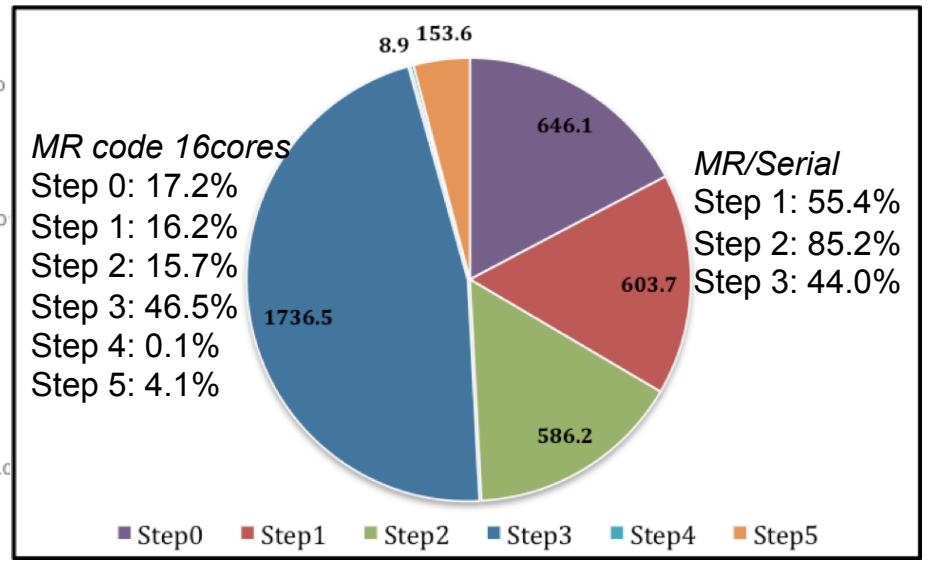


Comparison of original and ported code









Conclusions

- It's not essential to comprehend the entire algorithm to be able to port codes
  - Complexities with data partitioning, scheduling, handling machine-failures and communication are automatically handled by the framework
- Skewed data can have a significant impact on performance of Hadoop
  - Essential to understand data-access patterns to be able to modify the algorithm to operate on well-distributed key/value pairs, and to lessen the need for global synchronisation across all reduce tasks
- Choosing the right key is the key!

- *Future work*
  - *Performance comparison on EDIM1, UK HE clouds & AWS/EMR up to 128+ cores*
  - *Porting of ccc-gistemp to other scalable systems intended for data-intensive computing such as Dryad, All-Pairs and Pregel*
  - *Performance evaluation of the available key/value stores such as Voldemort, HBase, PostgreSQL and Redis*
  - *Investigate implementations of MapReduce utilising high-performance filesystems and key-value store based MapReduce*