A Software Package for Automated Partitioning of Catchments

Ralf Denzer, Tobias Kalmes, Udo Gauer

A Software Package for Automated Partitioning of Catchments

Ralf Denzer, Tobias Kalmes, Udo Gauer

Environmental Informatics Group (EIG), Saarbrücken, Germany
ralf.denzer@enviromatics.org

Abstract. This paper reports about a software package which has been developed to automatically partition hydrological networks (catchments) into clusters of similar size. Such clustering is useful for parallel simulation of catchments on distributed computing systems and is typically based on heuristic graph algorithms.

There have been a few approaches to automatically partition catchments, but literature research indicates that there seems to be no systematic investigation of the usefulness of different graph algorithms for catchment partitioning over a reasonable number of real world data sets. Our study aims at making a step in this direction.

The paper describes the software package, which has been implemented in Java, its pluggable architecture, and initial experiments using the European catchment dataset ECRINS. The paper presents work in progress.

Keywords: parallel simulation · hydrological network · graph clustering

1 Introduction

Some computational problems of river catchment simulations require large amounts of computing time. It is therefore just natural to aim at parallelizing such computations. At the core of any parallelization, the first question is whether an algorithm can be parallelized and how. As catchment simulations are dynamic flow problems, both spatial and temporal subdivision is an option.

A look at the literature suggests that while there have been several proposed solution, a systematic study is lacking. The aim of the work presented in this paper is a long term one: to start with a systematic approach coping with the parallelization of the problem (which means the partitioning of the catchments here) and to end in computational infrastructures for easy deployment of parallelized models.

In order to approach the catchment partitioning problem systematically, we chose to apply a divide and conquer strategy, starting with the most simple problem setup. The experiments reported in this paper are based on the following preconditions:

1. Precondition 1: the catchment is represented by a binary graph.
2. Precondition 2: the algorithm to be parallelized is the same in every node.

It is clear that in reality other cases may occur: the graph may contain cycles and the algorithms may be different for different parts of the catchment, but many applications in the literature satisfy the above requirements anyway and binary graphs are common representations for catchments.

For the purpose of this study we have also completely ignored the question when the parallelization of a problem will start to pay off, depending of the compute time per time step per node compared to the amount and latency of data transfer between computing nodes. We were only interested in the first step, the automatic partitioning.

A *binary graph* is a special form of a directed acyclic graph (DAG). Each graph node has a maximum of two predecessor upstream nodes, and a maximum of one downstream node. The problem of our study is defined as follows:

For a binary graph **G**, find a clustering

$$\mathbf{C} = \{ \mathbf{S_i, i = 1,n} \} \tag{1}$$

into **n** clusters, where $\mathbf{S_i}$ are binary graphs and

$$\cup \mathbf{S_{i,\ i=1,n}} = \mathrm{G} \tag{2}$$

,which means that the clusters taken together represent the graph **G**. Let **|G|** be the graph norm (number of nodes) in a binary graph. An optimal clustering $\mathbf{C_{opt}}$ is one which satisfies the following equation:

$$|\mathbf{S_i}| = |\mathbf{S_j}|, \text{ for all } \mathbf{i,j} \tag{3}$$

Equation (3) means that for the purpose of parallel computing, it is desired that all clusters have equal size. This reflects precondition 2.

2 Space and Time Parallelization

A dynamic flow problem can be parallelized in space and time, as long as the flow is represented by a DAG in which downstream nodes only receive input from upstream nodes. There are two principle possibilities to use a distributed computing infrastructure for parallelization: a) a large catchment is split spatially into several catchments, which are computed in parallel, and b) for different time steps or time periods there is a different processor per node. Theoretically one could assign **|G|** * T (T being the number of time steps) processors to the computing problem, which would however result in unrealistically large hardware needs. In reality it makes sense to allocate one computing node for a clustering of nodes, to compute a time period of reasonable size and then to communicate a partial time series between the clusterings through connecting nodes, in order to balance the trade-off between using multiple processors (positive influence) and communication delays and latencies (negative influence).

The long term interest of this study is to get a better understanding of the different trade-offs, based on a large number of real world datasets, and not based on very few

(or only one) dataset or very small (or even toy) datasets. For this purpose it is necessary

1. to spatially cluster many different datasets, including large datasets with many nodes, in order to gain experience how good the spatial partitioning is in reality – for this step a concrete model is not needed,
2. to actually run many model runs based on these clusterings in a distributed environment and measure their performance (which will be quite an effort and will require a lot of automation), or
3. to try to build a theoretical model of the distributed computing system based on execution per time step, average latencies and communication overhead, or
4. to experiment with a dummy model on a real distributed computing system

To this end we have started with the first step, and have implemented a software suite for spatial partitioning (or clustering). Along with the core software package, three published algorithms and one new algorithm have been implemented which we have used for the first batch of partitioning experiments.

It is our intention to carry on with the first step with as many datasets as we can get hold of and to implement more algorithms in the future. We also intend to conduct experiments according to step 4. We are also investigating how to move the clustering into a cloud infrastructure in order to provide a dynamic and scalable clustering service.

It should be noted though that this study is carried out with groups of Masters students only, without any external funding sources and support.

3 Related Work

The literature base on partitioning of catchments is not large but very diverse, approaching the problem from different angles. A 2001 article by M. Grübsch and O. David [1] gives a good introduction into the problem, its computational complexity and graph properties which may be taken into account when developing partitioning algorithms. They also propose a heuristic algorithm as a solution. In [2], a generalised computational architecture is proposed which allocates computing nodes over a master-slave pattern using load balancing, which is a standard approach in distributed computing. In [3], a SWAT model is parallelized but it remains unclear how the subcatchments have been partitioned. In [4] the authors state that a more generalized approach based on well understood software patterns would help practitioners develop parallel simulation solutions. One commonality of published papers is that whatever is proposed, the examples included in published studies have been small sets of case studies. Our study intends to start a systematic investigation over large sets of cases, starting with the partitioning problem.

4 The Software Package

The software implementation uses a pluggable architecture. At the core is a class called `ClusteringSuite` which dynamically manages a clustering run (see fig. 1). It loads clustering algorithms and test criteria (algorithms computing some quality measure of the generated clusters).

A new clustering algorithm or a new test criterion respectively only has to implement a simple interface and will be dynamically loaded based on entries in a property file denoting the algorithms and test criteria to be used. The suite dynamically loads algorithms via a method called addAlgorithm() add test criteria via a method called addTestCriterion().

```
class ClusteringSuite {
  …
  addAlgorithm() { load and add an algorithm }
  addTestCriterion() { load and add a test criterion }
  …
  List<ExperimentBundle<String>> doCluster {
  for all samples
      for all algorithms
        call clustering algorithm for sample
          for all test criteria
            calculate criterion
        save clustering result in json file
}
```

Fig. 1. Parts of the clustering suite interface

The current implementation provides the following test criteria as built-in elements:

- AverageSize — gives the average size of the clusters
- AboveAverageSize — counts how many clusters are above average
- BelowAverageSize — counts how many clusters are below average
- MedianSize — gives the median size of the clusters
- AverageWeight — gives the average weight of the clusters
- MedianWeight — gives the median weight of the clusters
- DeltaMedianAverageSize — gives the delta of average size and median size
- Correctness — determines whether the algorithm is correct at all

The package contains two different executable programs, one called `Clusterer` and one called `ExperimentController`. While the clusterer will cluster one set of samples given by the user (respectively passed to the program), the experiment controller will generate samples which cluster the same catchment into many different sizes. This is for automation of the clustering experiments.

5 The Experiment

5.1 The Catchment Dataset

For the first study we have used the ECRINS[1] dataset, which can be downloaded from the web site of the European Environment Agency[2]. The dataset contains amongst others a European-wide hydrological network with approximately 1.2 million river segments. The download section of the EEA site contains several datasets. In order to access the hydrological network data, the "EcrRiv" dataset is needed. The dataset is well documented in [5]. For the experiments we have chosen a variation of catchment size from 1500 nodes to nearly 1460000 nodes (see table 1). The *river id* is the column *river_id* in table *c_tr* of ECRINS and the *end node* is the draining node (column *nodid* in table *c_node* of ECRINS). This data was run against several algorithms.

River	River id	End node	Nodes
Volga	Z_C0000603	Y000601274	146585
Danube	Z_C0000897	Y000828008	114318
Kama	Z_C0000605	Y000312116	64024
Dnieper	Z_C0000089	Y000728340	42962
Don	Z_C0000631	Y000640649	32398
Rhine	Z_C0000823	Y000617080	25270
Po	Z_C0000025	Y000962481	23074
Ural	Z_C0000588	Y000537580	21391
Ebro	Z_C0000053	Y001234399	15648
Douro	Z_C0000049	Y001129129	12459
Drave	Z_C0000933	Y000912526	11966
Loire	Z_C0000067	Y000774881	9815
Adige	Z_C0000088	Y000952540	7608
Isere	Z_C0000750	Y000956258	6954
Adda	Z_C0000020	Y000955093	3385
Moselle	Z_C0000899	Y000660011	3054
Neckar	Z_C0000896	Y000693135	1578

Table 1. Catchments used in initial experiments

[1] http://www.eea.europa.eu/data-and-maps/data/european-catchments-and-rivers-network
[2] http://www.eea.europa.eu/

5.2 Clustering Algorithms Used

After literature research, 3 algorithms were chosen for the implementation of the first experiments. The choice of algorithms was not easy, as many proposed graph clustering algorithms cope with more general graphs, and the documentation is not always clear and detailed enough to use it as a blueprint for implementation. As a start, we decided to implement three algorithms: *K-Means* [6], *RNSC* [7] and *Spectral Clustering* [8]. During the course of the investigation, an idea lead to the implementation of a new algorithm called *Neighbourhood Clustering*. This algorithm was also used in the tests and may be published at a later stage after more experimentation.

5.3 Aims

If you want to distribute a catchment simulation over a distributed homogeneous computing infrastructure, it is desirable that all clusters have the same size. Then different computing nodes would not have to wait for each other when they are passing messages between clusters between time steps (or time periods) simulated. In reality this is never achievable with cluster size greater than 1 (one computing node per graph node), because the binary graphs representing real world catchments are highly unbalanced. Real results have clusterings in which clusters are not of equal size.

It was our goal to achieve clusterings where the maximum cluster size and the minimum cluster size do differ only *by a factor of 2 to 2.5*. The rationale behind this aim is that due to the asynchronous nature of operating systems, network communication, latencies and so forth the overall runtime behaviour of a distributed simulation is not predictable anyway, and this aim seemed reasonable, achievable and still practical for distributed simulation. In order to make the results more visible we define the *maximum spread* as that factor (*maximum spread := maxsize / minsize*), and the *spread curve* (figure 2) as a curve which shows the spread for all generated clusters, where the clusters are ordered by size from left to right.

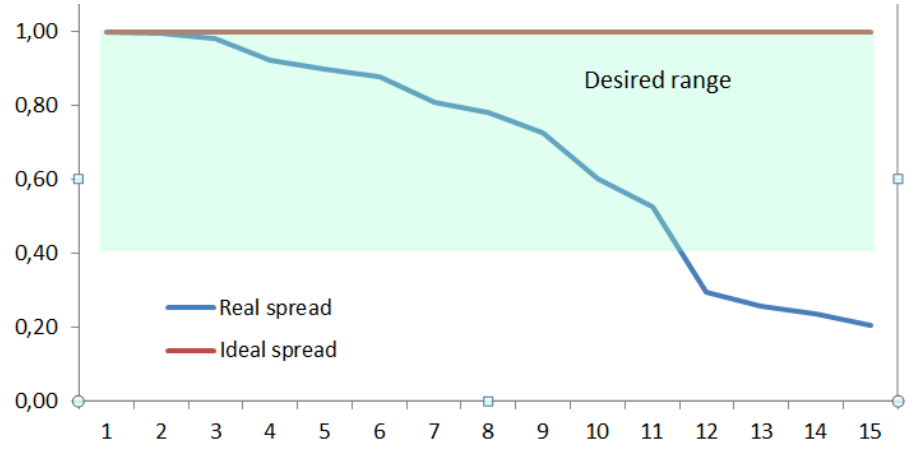

Fig. 2. Real vs. ideal spread curve (Neckar, 1500+ nodes, NH algorithm, 15 clusters)

5.4 Experiment Setup

In our initial experiments we ran all catchments against several algorithms where it was practical. Spectral clustering, for instance, was not used in all cases as it has considerable runtime requirements. Also clustering the largest catchments is a problem of run-time om standard machines. For each catchment in the experiment, the *desired cluster count* was chosen, starting with approximately 1/100 of the catchment size and iterating to approximately 1/10 of the catchment size with different step sizes, depending on the size of the catchment (for instance for river Neckar with 1578 nodes, an iteration was performed from desired cluster size 15 to 150, for each integer in between. These initial experiments were aimed at getting a first idea what might be achievable with the algorithms chosen.

6 Discussion

Our initial findings are that not all results are in the order of magnitude which we had hoped. Some clusterings have a very large variation in number of nodes. The best spreads are in the order of magnitude of 2.5, but particularly the very large catchments seem to be difficult to cluster evenly.

Some algorithms produce stray nodes, clusters of very small size – in some cases many of them. It is not clear yet whether those can be easily re-connected to other clusters (or to each other) in post-processing without unbalancing the quality of the solution again.

We have also not quality-checked the implemented algorithms over a large number of experiments, and a real problem is that there are no datasets in the literature to which we could compare our results to.

Also the choice of starting points influences the clustering, and some algorithms do have parameters which influence their behaviour. We have not yet done any variation over the automatically chosen starting points and over these parameters.

The computation time for the clusterings is another problem. It is just not practical to run many clustering runs over reasonable size clusters on standard machines, as some algorithms (particularly Spectral Clustering) just need too long to compute one clustering. Therefore particularly the large catchment have not been run over all variations of algorithms and cluster sizes which we had initially planned.

Therefore, in another project course in 2014/2015, we plan to extend our solution to a cloud infrastructure, in which we aim at running many more variations of experiments in order to find better solutions.

Acknowledgements

This experiment was carried out as part of a Master level project course in 2013/2014, with a group of 6 students, of which 2 agreed to co-author this paper. The resulting software was cleaned and refactored by the main author after the course had produced

the first results. It is our intention to continue these investigations and we welcome collaboration with external partners.

References

1. M. Grübsch, O. David, How to Divide a Catchment to Conquer Its Parallel Processing, an Efficient Algorithm for the Partitioning of Water Catchments, Mathematical and Computer Modelling 33 (2001), 723-731
2. H. Wang et al., A common parallel computing framework for modeling hydrological processes of river basins, Parallel Computing 37 (2011), 302–315
3. S. Yalew et al., Distributed computation of large scale SWAT models on the Grid, Environmental Modelling & Software 41 (2013) 223-230
4. R. Denzer, P. Fitch, I. N. Athanasiadis, D. P. Ames, Parallel simulation of environmental phenomena, International Congress on Environmental Modelling and Software 2014, http://www.iemss.org/sites/iemss2014/proceedings.php, ISBN: 978-88-9035-744-2, 2014
5. EEA, EEA Catchments and Rivers Network System, ECRINS v1.1, EEA Technical report No 7/2012, European Environment Agency, 2012, ISSN 1725-2237
6. Kanungo T., Mount D. M., Netanyahu N. S., Piatko C. D., Silverman R., Wu A. Y.: An efficient k-means clustering algorithm: Analysis and implementation. In: IEEE Trans. Pattern Analysis and Machine Intelligence. 24, 2002, S. 881–892. doi:10.1109/TPAMI.2002.1017616. Abgerufen am 24. April 2009.
7. A. D. King, Graph Clustering with Restricted Neighbourhood Search, Thesis, http:// citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.129.2497&rep=rep1&type=pdf, 2004
8. A. Y. Ng, M. I. Jordan und Y. Weiss, On spectral clustering: analysis and an algorithm, http://snap.stanford.edu/class/cs224w-readings/ng01spectralcluster.pdf