DEVELOPING APACHE SPARK BASED RIPLEY'S K FUNCTIONS FOR ACCELERATING SPATIOTEMPORAL POINT PATTERN ANALYSIS

Z. Gui [1, *], Y. Wang [2], Z. Cui[2], D. Peng [1], J. Wu [1], Z. Ma [1], S. Luo [1], H. Wu [2]

[1] School of Remote Sensing and Information Engineering, Wuhan University, Wuhan 430079, China - (zhipeng.gui, pengdh, wyw1294, zhipengma, luoshiqi)@whu.edu.cn
[2] State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University, Wuhan 430079, China - (yuan.wang, zousen_cui, wuhuayi)@whu.edu.cn

Commission V, WG V/4

KEY WORDS: Point pattern analysis, Visual analytics, Spatial agglomeration, High performance computing, Spatiotemporal index, Caching, Spatiotemporal data partitioning, Spatiotemporal object serialization

ABSTRACT:

Ripley's K functions are powerful tools for studying the spatial arrangement or spatiotemporal distribution characteristics of geographic phenomena and events in spatial analysis and has been used in many fields. However, the K functions are compute-intensive for point-wise distance comparisons, edge correction and simulations for significance test. Although parallel computing technologies have been adopted to accelerate K functions, previous works haven't extended the optimization from space to space-time dimension. This study presents an acceleration method for K functions upon state-of-the-art distributed computing framework Apache Spark, and four optimization strategies are leveraged to simplify calculation procedures and accelerate distributed computing respectively, including 1) spatiotemporal indexing based on R-tree with Sort-Tile-Recursive (STR) algorithm for reducing distance comparison when retrieving potential spatiotemporally neighbouring points; 2) Hash-Table-based caching for spatiotemporal edge correction weights reuse and reducing repetitive computation; 3) Spatiotemporal partitioning using KDB-tree as well as cylinder intersection redundancy strategy for decreasing ghost buffer redundancy in partitions and supporting near-balanced distributed processing; 4) Customized serialization of spatiotemporal objects and indexes for lowering the overhead of data transmission. Experiments verify the effectiveness and time efficiency of the proposed optimization strategies, and also evaluate the overall performance and scalability. Based on the proposed methods, a web-based visual analytics framework has been developed and publicly shared through GitHub, and four types of the distributed K functions are implemented, including space, space-time, local and cross K functions, which demonstrates its value on promoting geographical and socioeconomic studies.

1. INTRODUCTION

Effective approaches for detecting and studying the spatial arrangement or spatiotemporal distribution characteristics of geographic points would be helpful to investigate and interpret the spatiotemporal point process hidden behind geographic phenomenon or events (Cui et al., 2017). Among the approaches of point pattern analysis in spatial analysis, Ripley's K function (K function for short) is a multi-distance and scale-independent point pattern analysis method, so Modifiable Areal Unit Problem (MAUP) can be avoided (Hohl et al., 2017; Wang et al., 2020). Meanwhile, its parameters can be derived from study area, not like the bandwidth in kernel density estimation (KDE) that usually relies on experience (Yuan et al., 2019). The neighbors within the maximum distance are all considered, hence the information behind the point pairs can be fully utilized. Many variants of the K function have been developed for different analysis scenarios, including space-time, local, cross and network K functions, and there have been desktop-based software packages that provide K functions and its extensions (e.g., Spatstat, Splancs, Stpp in R). Therefore, K functions have been widely applied in many fields, such as ecology (Hendricks et al., 2017), archaeology (Winter-Livneh et al., 2010), epidemiology (Hohl et al., 2016), criminology (Pandit et al., 2016), sociology (Fu et al., 2017), economics (Kosfeld et al., 2011; Tian et al., 2017; Chen et al., 2018) and, biology and medical science (Sporring et al., 2019).

Although K function is a powerful tool in point pattern analysis, it also incurs computational challenges on spatiotemporal big data (Yang et al., 2015). K function is compute-intensive and become extremely time-consuming when data volume increases for several reasons: 1) Time complexity of pairwise distance comparison between all points is quadratic; 2) Weight calculation is need for point pairs to correct edge effect, which is positively correlated to the geometric complexity of the study area boundary; 3) A fair amount of simulations are demanded to conduct confidence evaluation for significance level of point pattern. The expected time cost of K function would be even higher when extended from spatial dimension to spatiotemporal dimension. As the result, the time efficiency of the classical desktop-based packages is far from satisfying for large data volume. It affects the user experience of geoprocessing significantly (Hu et al., 2019) and impedes further application. Hence, the acceleration of K functions is urgent to enable efficient spatiotemporal point pattern analysis.

High Performance Computing (HPC) technologies have been applied to tackle the compute-intensive challenges brought by spatial analysis (Guan et al., 2011; Gui et al., 2015). Multi-CPU (Zhang et al., 2016) and massive-GPU (Tang et al., 2015) methods have been developed to accelerate space K function for large point datasets. Although great achievements have been

* Corresponding author

made to accelerate the computing process of K functions, existing studies mainly focus on spatial dimension, and temporal dimension is seldom involved. These implementations are limited in scalability and workflow optimization due to relative expensive programming cost of the parallel frameworks. Without framework-level supports and abundant third-party libraries, advanced spatial extensions, fault-tolerance mechanism and spatiotemporal-aware scheduling are hard to be achieved. Distributed frameworks like Apache Hadoop and Spark are gaining ground in big geospatial analytics [24,25]. However, existing parallel optimization methods of K functions couldn't fit well in such a distributed data pipeline, and a systematic parallel method for K functions is highly desired.

To address these issues, this study adopts four generic optimization strategies for distributed K functions proposed by our previous study (Wang et al., 2020). Specifically, 1) spatiotemporal indexing is adopted to avoid unnecessary pair-wise comparison in point pair acquisition; 2) weight cache is designed to reuse spatiotemporal weights and decrease repetitive calculations; 3) spatiotemporal partitioning is used to balance workloads and communication overheads among computing nodes in the cluster; 4) customized serialization for spatiotemporal objects and indexes is developed to lower the overhead of data transmission between nodes. The performance experiments verify the efficiency of the proposed method.

The paper is organized as follows: Section 2 presented the proposed optimization methods for K functions. Section 3 analysed the performance and scalability of the algorithms through a group of experiments. Section 4 introduced the technical implementation of the developed web-based visual analytics framework. Section 5 drew conclusions.

2. METHODOLOGY

To lower the computational barrier of spatiotemporal point analysis for large datasets, we adopted a distributed computing mode of K functions over the distributed computing framework Apache Spark as shown in Figure 1. In this framework, multiple workers (i.e., computing node) conduct the divided computing tasks of K functions according to their local data storage, while the master node allocates tasks and gathers the processing result. When the driver program is submitted to the master, computing tasks will be generated and computing resources for the job will be allocated according to the submitted K function and Apache Spark parameters respectively. Apache Spark parameters include number of executors, CPU resources, and memory resources; while K function parameters include the type of the specified K function, JAR package of the algorithm, point dataset identifier, study area, spatial and temporal distance thresholds, edge correction method, simulation method, number of simulations and etc. Built-in spatiotemporal partitioner in master node handles data partition among workers. Customized serializer provides compact representation of spatiotemporal objects for reducing data transmission. The executors in the workers execute multiple threads and handle the assigned calculation tasks simultaneously. In each executor, an embedded spatiotemporal index builder indexes local data partition. Weight cache, spatiotemporal objects, and spatiotemporal index for local data are cached in partitions of the executor for accelerating calculation. The results will be transferred to and aggregated by the master when all the tasks are finished. Through the master-slave programming model, the distributed implementation of K functions can be scheduled, accelerated and monitored.

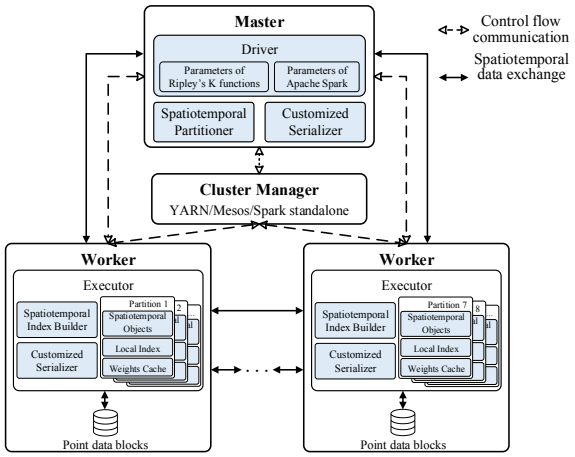


Figure 1. Master-slave mode of distributed K functions

A generic calculation workflow of distributed K functions that adopts the four strategies is shown in Figure 2 by taking space-time K function as an example (Wang et al., 2020). In the main procedure, both observed and simulated points are spatiotemporally partitioned before the calculation to balance workload and reduce IO overhead among computing nodes. Spatiotemporal indexes are built on each partition to boost neighbouring point query, and cache is utilized to reuse weights for spatiotemporal edge correction. In addition, customized serialization enables compact data transmission between nodes for spatiotemporal objects and indexes.

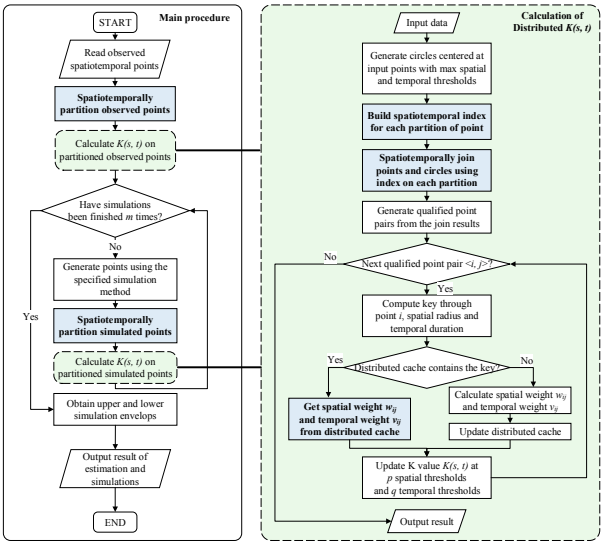


Figure 2. Optimized distributed calculation workflow by taking space-time K function as an example

2.1 R-tree-based Spatiotemporal Indexing

The calculation of K functions requires nested traversals on the points. The outer traversals cover every point, while the inner traversals only need to find the neighboring points that lie within the spatiotemporal thresholds ideally. To avoid unnecessary traversals, the point pair acquisition can be regarded as a query task, and spatiotemporal index might quickly narrow the query scope and decreases the comparison times of inner traversals. As the performance of range query the key in point pair acquisition, R-tree with Sort-Tile-Recursive (STR) algorithm [51] is adopted. As shown in Figure 3, points are bulk loaded into the tree and overlaps between the spatiotemporal MBR of nodes are avoided.

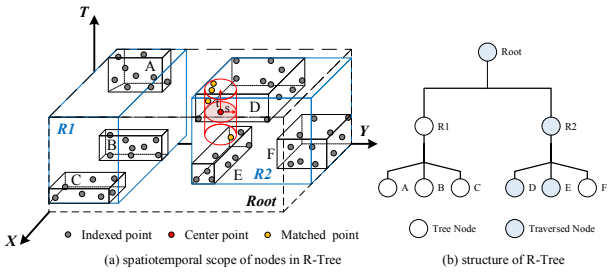


Figure 3. R-tree with Sort-Tile-Recursive (STR) algorithm for spatiotemporal point pair queries

In query stage, a series of cylinders centred at the spatiotemporal points (red point in Figure 3(a)) with spatial threshold as the radius and double temporal threshold as the height are constructed as query scope. Then the query scope will be compared with spatiotemporal cubes of the tree nodes. If they intersect with each other (light blue nodes in Figure 3(b)), same comparison operation will be performed on the child tree nodes, until the leaf nodes are reached. The points in leaf nodes will be directly compared with the query scope, and matched points (yellow points in Figure 3(a)) will be added to the result. Therefore, comparisons are made only for the nodes potentially matched the respective query.

2.2 Spatiotemporal Weight Caching

Edge correction weight calculation among the spatiotemporal point pairs is time-consuming, especially when the boundary of the study area becomes more complex. The repetition of calculation can be eliminated if there are multiple neighboring points having the same spatial distance and temporal distance from the same center point coordinate under the given spatial coordinate tolerance and spatial distance tolerance, or when conducing simulations using random permutation.

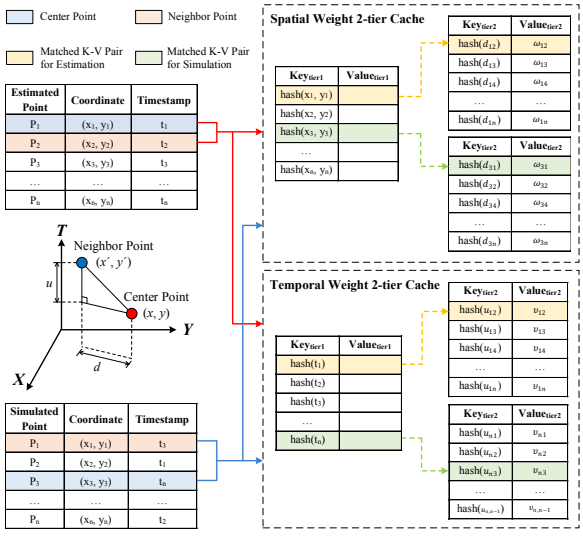


Figure 4. Cache for spatial and temporal weight reuse

To avoid repetitive calculation, spatial and temporal weights can be cached into two hash tables separately for reusing as illustrated in Figure 4. Spatial weight cache and temporal weight cache are filled with $< p_s, < d, \omega >>$ and $< p_t, < u, v >>$ entries respectively, where p_s and p_t are coordinate and timestamp of the center point, d and u are the spatial distance and temporal distance, and ω and v are the spatial weight and time weight. The key in the first-tier hash table is the hash value of p_s and p_t respectively, and the corresponding value is the second-tier table for p_s and p_t. While, the key of the second-tier hash table is the hash values of d and u respectively, and the corresponding value is the spatial and temporal weight.

2.3 KDB-tree-based Spatiotemporal Partitioning

Moderate data redundancy among data partitions is essential for alleviating unnecessary IO cost on data transmission between computing nodes, and eventually accelerating the computation. Partitioning using sample points can furtherly lower the computing cost for processing dataset with big data volume. According to relevant research, 1% of the samples are sufficient to obtain high quality partitions (Eldawy et al., 2015). Therefore, spatiotemporal partitioning is accomplished by building KDB-tree-based index using sample data to accelerate spatiotemporal scopes generation, as shown in Figure 5. 1) After loading the spatiotemporal points from storage system, the points are randomly sampled and sent to the master; 2) a KDB-tree-based spatiotemporal index is built on the sample points by the master. The number of partitions is decided by the indexing through construction parameters for the tree, such as maximum number of child nodes, maximum number of items; 3) Once the index is delivered to the workers, the workers query the leaf node to which each point belongs and build key-value pair $< id, p >$, where id is the unique identifier for spatiotemporal envelop of the leaf node, and p denotes the spatiotemporal point; 4) distributed points is repartitioned according to the key of pair, and key-value pairs with the same key is divided into the same partition. Eventually, spatiotemporally partitioned points can be derived from the value of pairs.

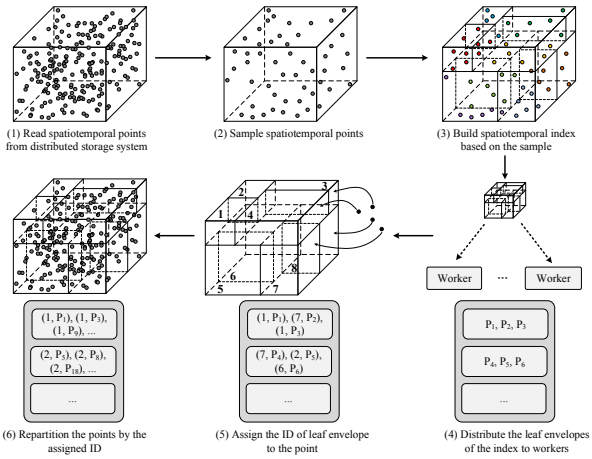


Figure 5. Process of spatiotemporal partitioning

2.4 Customized Serialization for Data Transmission

Big data processing frameworks, such as Spark, provide serializers with sufficient capabilities to handle simple objects, but for spatiotemporal objects, a compact representation is missing, which causes more bytes generated and transferred in the cluster. To solve this problem, customized serializations are developed for spatiotemporal objects including points, cylinders, envelopes and indexes. Here we take spatiotemporal cylinder as an example to compare the representation of default and customized serialization methods for the length limit, which shows a significant reducing in bytes in Figure 6.

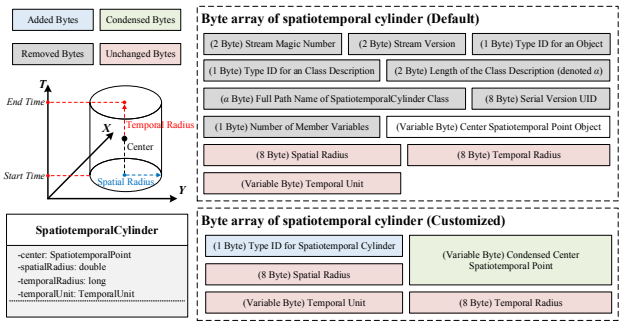


Figure 6. Representations of default and customized serializations for spatiotemporal cylinder

3. EXPERIMENTS

To prove effectiveness and adoptability of the developed method, we analyse the performance improvement of different distribute K functions. For experiments in Section 3.1 to 3.5, cross K function was used, while for Section 3.6 and 3.7, space K function and space-time function were selected. Experiments were conducted on a private cloud supported by Apache CloudStack built upon 6 physical nodes. Each physical node that works as the agent of this private cloud has 24 CPU cores of 2.4 GHz and 64GB memory. They are connected by local network with 1 Gbps. 9 VMs with 8 virtual CPU cores of 2 GHz and 16GB memory running CentOS 7.2 were created on the private cloud. 1 VM served as master, and the other 8 VMs served as workers were evenly hosted on 4 physical nodes. The versions of Spark and Hadoop are v2.3 and v2.7 respectively. The experiment datasets were resampled from the enterprises registration data in Chongqing, China (2,119,419 points in total) from year 1949 to year 2018 recorded by the bureaus of Administration for Industry and Commerce (AIC) of China after imputation (Li et at., 2018). For more performance analysis on space-time K function, please refer to our previous research (Wang et al., 2020).

In order to evaluate the performance of distributed K functions, speedup factor (SF) and acceleration factor (AF) are used to measure how much speedup of optimization strategies achieved and how much acceleration of the distributed system achieved respectively, as shown in formula 1.

$$SF = T_{original}/T_{optimized}$$
$$AF = T_{standalone}/T_{distirbuted} \quad (1)$$

where $T_{original}$ and $T_{optimized}$ are the execution time of the original and optimized K function respectively, while $T_{standalone}$ and $T_{distirbuted}$ are the execution time of the K function performed on standalone machine and distributed cluster respectively. All the execution time mentioned above excluded time spent on data input and output.

3.1 Performance of Spatiotemporal Indexing

Since spatial cross K function studies the spatial correlation between two group of points, two resampled subsets with equal number of points (n=50,000) of the experiment point data were used as the input point dataset pair in this experiment to compare the execution time of cross K function with spatiotemporal index and without index. The degree to which the times of inner traversals could be reduced depends on the maximum spatial distance s_{max}. Therefore, the proportion of spatial query scope to study area is set from 1/256 to 1/16 exponentially. In practice, the index building times are less than 0.1s, and almost has no influence on total execution time.

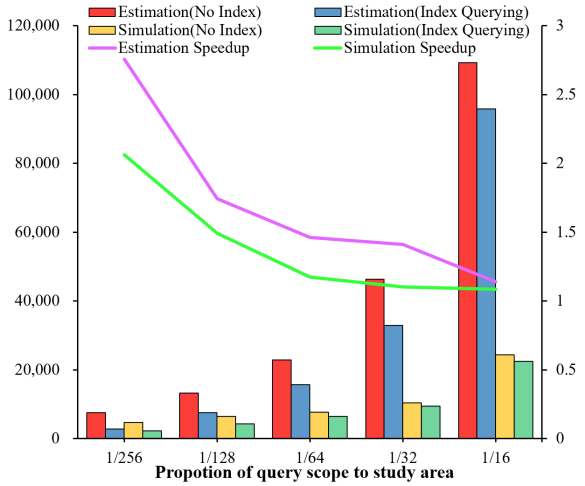


Figure 7. Performance comparison of cross K function with and without spatiotemporal index under increasing query scopes

The results in Figure 7 indicate that spatiotemporal index would increase the performance of estimation and simulations at relatively short spatial distance, while as the spatial distance increases the performance for indexing querying become less effective and even worse than the solution without indexing. The speedup factor was higher than 1 when s_{max} was less than 1/16 of smaller side length of spatial boundary. It could be explained that the query scopes at higher s_{max} were not partial enough, then most of the tree nodes would be traversed and the index would lose its effect. While, at lower s_{max}, the speedup factor for estimation and simulation could reach 2.75 and 2.06 respectively, because the query scope would only interest with a small amount of tree nodes in the index. Meanwhile, point distribution has huge impact on the effectiveness of index. When the dataset is extremely skewed that most of points concentrated in certain peak areas, the spatiotemporal query scope will overlap with large proportion of R-tree nodes in index, and hence weaken the filtering capability of the index seriously.

3.2 Performance of Weight Caching

As the time complexity of spatiotemporal isotropic correction method is linear correlated to the complexity of boundary, this experiment was performed on two resampled subsets with equal number of points (n=50,000) at the same maximum spatial distance but with boundaries composed of different number of vertexes. The execution time of cross K function with cache was compared to that without cache.

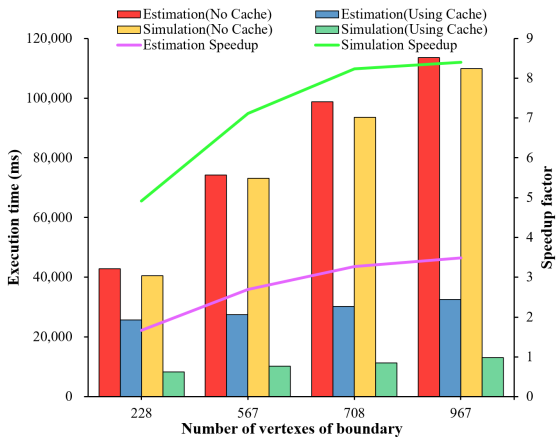


Figure 8. Performance comparison of cross K function with and without weight cache under increasing complexity of boundary

Figure 8 shows that the cache can eliminate the influence brought by the complexity of boundary. Execution times of estimations and simulations are independent with the boundary when using the cache, while the execution times of the counterpart solution without cache do increase linearly with the number of vertexes of the boundary. However, when the boundary is extremely simple, the calculation of weights could be quickly finished, while the overhead of cache caused by resizing and hash collision will offset the benefits of cache. Meanwhile, due to the difference in cache utilization, the speedup factor for simulation is higher than that of estimation.

3.3 Performance of Spatiotemporal Partitioning

This experiment was conducted on two resampled subsets with equal number of points (n=50,000) at the same maximum spatial distance with the same boundary using varying number of partitions. The execution time of cross K function with spatiotemporal partitioning was compared to that with hash partitioning, the default method in Apache Spark, to investigate the effect of spatiotemporal partitioning.

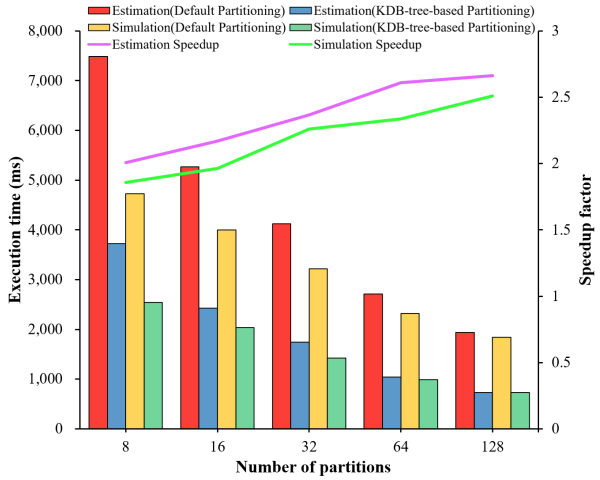


Figure 9. Performance comparison of cross K function with and without partitioning under increasing number of partitions

Figure 9 illustrates that spatiotemporal partitioning could avoid unnecessary data redundancy and accelerate calculation. Few partitions lead to less data redundancy but bring longer average execution time, while more partitions result in fine-grained task scheduling but also introduce more data transmission. Therefore, optimized partitioning needs to leverage execution time and transmission time by carefully adjusting the number of partitions.

3.4 Performance of Customized Serialization

The volume of data transmission in distributed systems is mainly related to the size of point data. Therefore, this experiment was conducted on datasets with different number of points under the same spatial distances with the same boundary. The execution time of cross K function with default serialization was compared to that with customized serializer to investigate the effect of customized serializer (Figure 10).

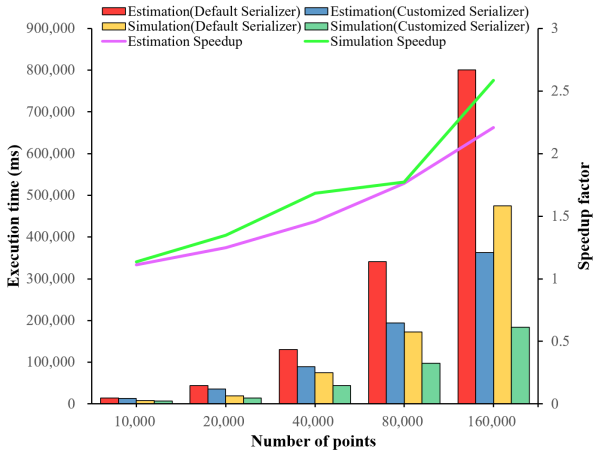


Figure 10. Performance comparison of cross K function with default and customized serializer under increasing point number

As is shown in Figure 10, customized serialization could reduce overhead of data transmission and improve performance of cross K function. The speedup factor was consistently higher than 1 and grew up with the increasing number of points since more points would generate more spatiotemporal objects. Customized serialization can archive more than 20 times of compression ratio, in turn archive more than 10 times and 40 times of speedup ratio for serialization and deserialization.

3.5 Performance of Integrated Four Optimization Strategies

The overall effectiveness of the four optimization strategies was evaluated by comparing execution time of cross K function with and without any procedure optimizations. The original algorithm and optimized algorithm are both Spark-based and tested on the same clustering computing environment with 8 worker nodes. The experiment was carried out on different size of point datasets. The maximum spatial distance threshold was 100km, which is 1/5 to the smaller side length of study area.

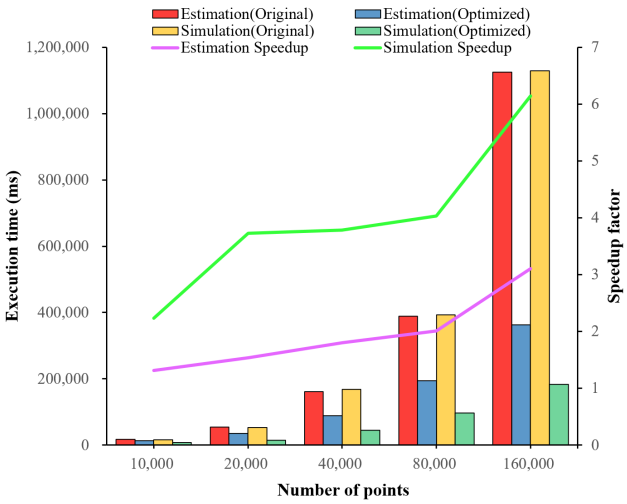


Figure 11. performance comparison with and without distributed optimization under increasing data volume

Figure 11 demonstrates that the speedup factor achieved by four optimization strategies increases as data size increases. Although the setting of the maximum spatial distance threshold and extremely skewed point distribution in this experiment make the spatiotemporal index almost noneffective on time efficiency improvement as discussed in section 3.1, the estimation still achieved about 2.0 times and 3.1 times speedup at data size

80,000 and 160,000 respectively. Meanwhile, the speedup for simulation was higher than that for estimation on large data sizes because of less calculation for spatiotemporal weight.

3.6 Scalability Analysis

The scalability of distributed K functions was evaluated by comparing performance of estimations and simulations on different number of nodes in the cluster. In this experiment, we take the space K function and space-time K function as examples. The experiment was performed on a resampled point dataset (n=200,000) at spatial distance from 0 to 20km and temporal distance from 0 to 20 months with 1 km as spatial step and 1 month as temporal step.

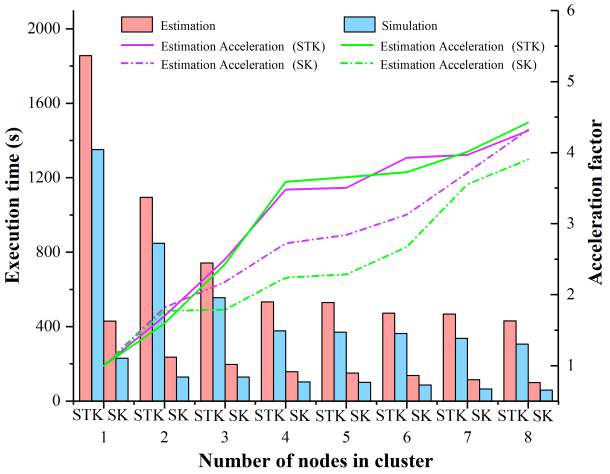


Figure 12. Performance analysis of space K function (denote as SK) and space-time K function (denote as STK) under increasing number of nodes in cluster

As shown in Figure 12, with the increasing of the computing nodes, the execution times of space K function and space-time K function are reduced and the acceleration factors increase in general. The execution time of space K function is shorter than that of space-time K function at all node scales (i.e., number of computing nodes), but the AF trend of space K function is similar to that of space-time K function. As the node increases, the AF values of both space K function & space-time K function tends to be converged. The reason is that the performance improvement brought by the increase of computing resources is gradually offset by the increasing cost of network communication. With the increase of the computing node, the impact of computing resources on execution time is gradually reduced, which cannot improve execution efficiency anymore eventually. Further optimization would be made by improving the utilization rate of computing resources and reducing the network overheads introduced by small partition size.

3.7 Overall Speedup Analysis

This experiment analyses the overall performance of the distributed K functions that integrate four optimization strategies and benefit from powerful computer resources provided by the cluster. Here, we select space K function as a case study. Since the datasets used in the experiments of Section 3.1 to 3.5 are relatively small and can't demonstrate the power of the distributed K functions, this experiment was performed on a group of resampled point datasets with increasing number of points from 10,000 to 300,000. Same to the scalability analysis, the spatial distance increases from 0 to 20km with 1 km as spatial step. The optimal number of partitions were inferred and specified from partitioning experiments. The execution time of the K function on 8 nodes with optimization strategies were compared to that on standalone computer with and without adopting optimization strategies bothly to investigate the overall speedup.

Table 1 shows overall speedup by adopting the proposed optimization strategies and Apache Spark clusters comparing with the original algorithm ran on single VM with Spark local mode. We can find that the overall speedup increases as the data size increases significantly, which can achieve 76.23 times and 92.4 times for estimation and simulation at data size 300,000 respectively. The execution times of the optimized algorithm ran on 8 worker nodes grown slowly as the increase of data size and didn't show the exponential increase as that of the original non-optimized algorithm on single VM because of the adoption of four optimization strategies and clustering computing.

Extremely skewed spatiotemporal distribution of the dataset also has huge impact on acceleration effect. To demonstrate the extreme performance of our algorithm, an experiment for space-time K function was also performed on a point dataset generated by ArcGIS in our previous study (Wang et al., 2020). In that dataset, the points are random distributed in United States and a simplified administration boundary of United States with 85 points is used as the boundary of the study area. The overall acceleration factor under the same experiment environments as here with 8 computing nodes can even achieve more than 1000 times for estimation and simulation at data size 400,000. That is because, the adjacent point pair grows linearly under random spatiotemporal data distribution, as the result the index and partitioning can work effectively, and eventually gained significant speedup ratios.

Number of Points	Execution time (s)						Overall acceleration factor	
	Original on 1-node		Optimized on 1-node		Optimized on 8-nodes			
	Estimation	Simulation	Estimation	Simulation	Estimation	Simulation	Estimation	Simulation
10,000	145.918	86.556	152.338	58.224	28.454	12.053	5.13	7.18
20,000	303.028	160.409	259.941	101.899	38.038	14.738	7.97	10.88
50,000	922.567	487.231	629.254	239.067	48.146	20.127	19.16	24.21
100,000	2118.678	1158.916	903.555	402.863	59.765	27.056	35.45	42.83
200,000	5175.611	3745.507	1551.980	962.850	99.295	58.926	52.12	63.56
300,000	8942.642	6910.022	1840.657	1262.219	117.314	74.776	76.23	92.40

Table 1. Overall speedup of space K function by adopting optimization strategies and clustering computing

4. WEB-BASED VISUAL ANALYTICS FRAMEWORK

To promote the applications of the distributed K functions, a multi-tier web-based visual analytics framework was designed and implemented as shown in Figure 13. Four types of distributed K functions were implemented and integrated, including space K function, space-time K function, local K function and cross K function. The data storage tier comprises HDFS distributed file system for data management. The data processing tier is built upon Spark RDD API, spatial object API of JTS and GeoTools to support object representations and spatial operations. Spatiotemporal objects and Spatiotemporal RDD are defined to represent distributed spatiotemporal objects. Besides, the index builder, hash-table-based cache, partitioner and customized serializer are integrated. On the top of data processing tier, web service tier and data visualization tier are built to facility the invocation of distributed K functions and visual analytics through a loosely-coupled and web-based approach. The web service tier uses Jersey to provide RESTful services. In data visualization tier, the map interface is implemented upon Leaflet.js, Deck.GL, while the base map is from OpenStreetMap, and the thematic chart is supported by D3.js and Apache Echarts.

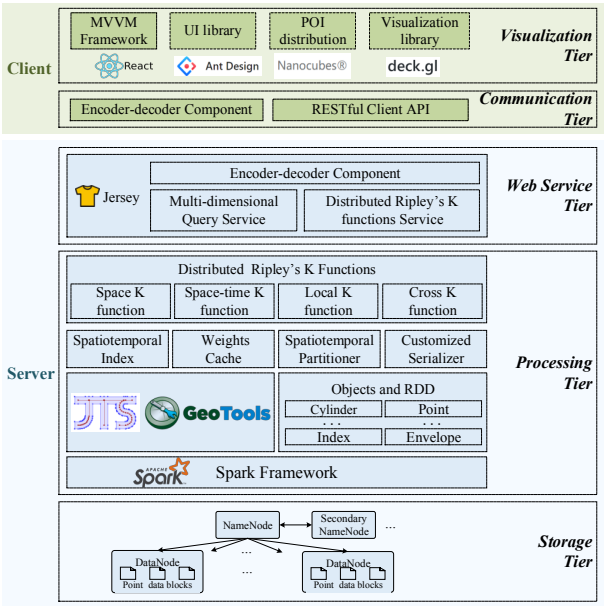
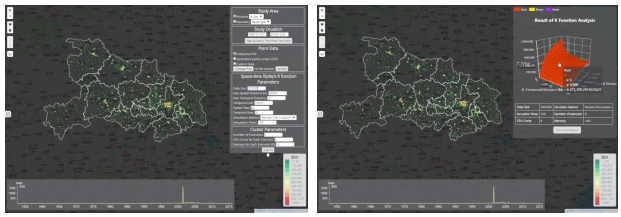
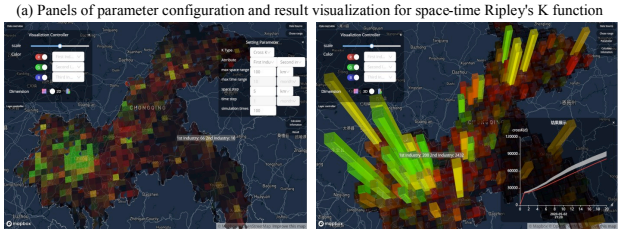


Figure 13. Web-based visual analytics framework for distributed Ripley's K functions

Through this framework, users can conduct efficient and flexible point pattern analysis by using distributed computing resources without knowing the underlying implementation details. The framework provides multi-dimensional and multi-granular filtering functions, including spatial filter (province, city, street), temporal filter (year, month, day), and customized categorical filter by using Nanocubes. As shown in Figure 14, the user can select the spatiotemporal points for analysis, specify the type of K functions, and configure both the K function parameters and the Spark cluster parameters. For K function parameters, the user can specify study area through either MBR or the identifier of the administrative boundary, simulation method, spatiotemporal distance thresholds and step sizes, which may vary in the type of K functions. For Spark parameters, besides the default configurations, the total number of executors, CPU cores and memory for each executor can be customized. When a user specifies and submits above parameters through parameter panels on the web pages, the job will be executed by the cluster and the running status can be checked through the status panel. When the calculation is finished, the respective results will be transferred back to the client and shown on the 2D/3D maps and plots for interaction (Figure 14). The calculation results can also be downloaded in tabular form through download button. The source code of the framework can be found on our GitHub repository (https://github.com/ZPGuiGroupWhu/Spark-based-Ripley-K-Functions).



(a) Panels of parameter configuration and result visualization for space-time Ripley's K function



(b) Panels of parameter configuration and result visualization for cross Ripley's K function

Figure 14. Graphical user interfaces (GUIs) of the Web-based visual analytics framework

5. CONCLUSION

This paper utilized four acceleration strategies of distributed K functions for better supporting point pattern analysis, including spatiotemporal-index-based point pair acquisition, cache weight reuse, spatiotemporal partitioning and customized serialization for spatiotemporal objects. Based on that, we developed web-based visual analytics framework upon Apache Spark, and four types of K functions were implemented for point pattern analysis. Experiments verified that the proposed method can reduce the time complexity of K functions for large datasets. Performance evaluations revealed that the first two optimization strategies upon calculation procedure could speed up the calculation with appropriated query scope setting even in a stand-alone execution environment; while the last two strategies accelerate the calculation furtherly by balancing workloads and decreasing data transmission overheads. This study may provide insight on how to lower the computing barrier of other spatial analysis methods and promote their broader applications for large datasets.

Future work would focus on supporting more K function variants and extending the proposed method from univariate homogeneous isotropic point analysis to bivariate, inhomogeneous and anisotropic point pattern analysis (Møller and Toftaker, 2014). Meanwhile, the developed web-based visual analytics framework will be furtherly enhanced by providing more visualization and interaction functions to better support human-computer interactive geovisual analytics in various spatial analysis applications.

ACKNOWLEDGEMENTS

This study is supported by National Key R&D Program of China (No. 2017YFB0503704 and No. 2018YFC0809806) and National Natural Science Foundation of China (No. 41971349, No. 41501434 and No. 41371372).

REFERENCES

Chen, Y., Qin, K., Gui, Z., Wu, H., 2018. Exploring spatial agglomeration of China's secondary industry based on registration data of industrial and commercial enterprises, *J. Liaoning Tech. Univ. (Natural Sci.),* 37, 602-610.

Cui, Z., Xie, G., Gui, Z., and Wu, H., 2017. Analyzing the Spatiotemporal Distribution of Different Industries in Wuhan City Using Enterprise Registration Data. *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci.*, XLII-2/W7, 5-10. doi.org/10.5194/isprs-archives-XLII-2-W7-5-2017.

Eldawy, A., Alarabi, L., Mokbel, M.F., 2015. Spatial partitioning techniques in SpatialHadoop. *Proc. VLDB Endow.*, 8, 1602-1605. doi.org/10.14778/2824032.2824057.

Fu, J.Y., Jing, C.F., Du, M.Y., Fu, Y.L., Dai, P.P., 2017. Study on adaptive parameter determination of cluster analysis in urban management cases. *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci*., XLII-2/W7, 1143-1150. doi.org/10.5194/isprs-archives-XLII-2-W7-1143-2017.

Guan, Q., Kyriakidis, P.C., Goodchild, M.F., 2011. A parallel computing approach to fast geostatistical areal interpolation. *Int. J. Geogr. Inf. Sci.*, 25(8), 1241-1267.

Gui, Z., Yu, M., Yang, C., Jiang, Y., Chen, S., Xia, J., Huang, Q., Liu, K., Li, Z., Hassan, M.A., Jin, B., 2016. Developing subdomain allocation algorithms based on spatial and communicational constraints to accelerate dust storm simulation. *PLoS One*, 11, e0152250. doi.org/10.1371/journal.pone.0152250.

Hendricks, K.E., Christman, M., Roberts, P.D., 2017. Spatial and Temporal Patterns of Commercial Citrus Trees Affected by Phyllosticta citricarpa in Florida. *Scientific Report*, 7, 1641. doi.org/10.1038/s41598-017-01901-2.

Hohl, A., Delmelle, E., Tang, W., Casas, I., 2016. Accelerating the discovery of space-time patterns of infectious diseases using parallel computing. *Spat. Spatiotemporal. Epidemiol.*, 19, 10-20. doi.org/10.1016/j.sste.2016.05.002.

Hohl, A., Zheng, M., Tang, W., Delmelle, E., Casas, I., 2017, Spatiotemporal point pattern analysis using Ripley's K function. *Geospatial Data Sci. Tech. Appl*., 155-175. doi:10.1201/b22052.

Hu, K., Gui, Z., Cheng, X., Wu, H., McClure, S., 2019. The Concept and Technologies of Quality of Geographic Information Service: Improving User Experience of GIServices in a Distributed Computing Environment. *ISPRS Int. J. Geo-Information*, 8, 118. doi.org/10.3390/ijgi8030118.

Kosfeld, R., Eckey, H.F., Lauridsen, J., 2011. Spatial point pattern analysis and industry concentration. *Ann. Reg. Sci.* 47, 311-328. doi.org/10.1007/s00168-010-0385-5.

Leutenegger, S.T., Lopez, M.A., Edgington, J., 1997. STR: A simple and efficient algorithm for R-tree packing. in: *Proc. 13th Int. Conf. Data Eng. IEEE*, 497-506.

Li, F., Gui, Z., Wu, H., Gong, J., Wang, Y., Tian, S., Zhang, J., 2018. Big enterprise registration data imputation: Supporting spatiotemporal analysis of industries in China. Computers, Environment and Urban Systems, 2018, 70, 9-23. doi.org/10.1016/j.compenvurbsys.2018.01.010

Møller, J., Toftaker, H., 2014. Geometric Anisotropic Spatial Point Pattern Analysis and Cox Processes. *Scand. J. Stat*., 41, 414-435. doi.org/10.1111/sjos.12041.

Pandit, K., Bevilacqua, E., Mountrakis, G., Malmsheimer, R.W., 2016, Spatial Analysis of Forest Crimes in Mark Twain National Forest, Missouri. *J. Geospatial Appl. Nat. Resour.,* 1(1), 39-53.

Song, Y., Gui, Z., Wu, H., and Wei, Y., 2017. A Web-Based Framework for Visualizing Industrial Spatiotemporal Distribution Using Standard Deviational Ellipse and Shifting Routes of Gravity Centers. *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci*., XLII-2/W7, 129-135. doi.org/10.5194/isprs-archives-XLII-2-W7-129-2017.

Sporring, J., Waagepetersen, R., Sommer, S., 2019. Generalizations of Ripley's K-function with Application to Space Curves. *Int. Conf. Inf. Process. Med. Imaging*, 731-742. doi.org/10.1007/978-3-030-20351-1.

Tang, W., Feng, W., Jia, M., 2015. Massively parallel spatial point pattern analysis: Ripley's K function accelerated using graphics processing units. *Int. J. Geogr. Inf. Sci*., 29(3), 412-439.

Tian, S., Wang, J., Gui, Z., Wu, H., and Wang, Y., 2017. A Case Study: Exploring Industrial Agglomeration of Manufacturing Industries in Shanghai Using Duranton and Overman's K-Density Function. *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci*., XLII-2/W7, 149-154, doi.org/10.5194/isprs-archives-XLII-2-W7-149-2017.

Wang, Y., Gui, Z., Wu, H., Peng, D., Wu, J., Cui, Z., 2020. Optimizing and Accelerating Space-Time Ripley's K Function based on Apache Spark for Distributed Spatiotemporal Point Pattern Analysis. *Future Generation Computer Systems*, 105, 96-118. doi.org/10.1016/j.future.2019.11.036.

Winter-Livneh, R., Svoray, T., Gilead, I., 2010. Settlement patterns, social complexity and agricultural strategies during the Chalcolithic period in the Northern Negev, Israel. *J. Archaeol. Sci*., 37, 284-294. doi.org/10.1016/j.jas.2009.09.039.

Yang, C., Sun, M., Liu, K., Huang, Q., Li, Z., Gui, Z., Jiang, Y., Xia, J., Yu, M., Xu, C., Lostritto, P., Zhou, N., 2015. Contemporary computing technologies for processing big spatiotemporal data. in: *Space-Time Integr. Geogr. GIScience*, 327-351. doi.org/10.1007/978-94-017-9205-9.

Yuan, K., Cheng, X., Gui Z., Li, F., Wu, H., 2019. A Quad-tree-based Fast and Adaptive Kernel Density Estimation Algorithm for Heat-map Generation. *Int. J. Geogr. Inf. Sci.*, 33(12), 2455-2476. doi.org/10.1080/13658816.2018.1555831.

Zhang, G., Huang, Q., Zhu, A.X., Keel, J.H., 2016. Enabling point pattern analysis on spatial big data using cloud computing: optimizing and accelerating Ripley's K function. *Int. J. Geogr. Inf. Sci*., 30(11), 2230-2252.