

Article

An Efficient Row Key Encoding Method with ASCII Code for Storing Geospatial Big Data in HBase

Quan Xiong ^{1,2} , Xiaodong Zhang ^{1,3,4} , Wei Liu ¹ , Sijing Ye ⁵ , Zhenbo Du ¹, Diyou Liu ¹ ,
Dehai Zhu ^{1,3,4}, Zhe Liu ^{1,3,4}  and Xiaochuang Yao ^{1,3,4,*} 

¹ College of Land Science and Technology, China Agricultural University, Beijing 100083, China; xiong@cau.edu.cn (Q.X.); zhangxd@cau.edu.cn (X.Z.); devilweil@cau.edu.cn (W.L.); tufangbobo@cau.edu.cn (Z.D.); diyouliu@cau.edu.cn (D.L.); zhudehai@cau.edu.cn (D.Z.); liuz@cau.edu.cn (Z.L.)

² Center for Spatial Information Science and Systems, George Mason University, 4400 University Dr, Fairfax, VA 22030, USA

³ Key Laboratory of Remote Sensing for Agri-Hazards, Ministry of Agriculture, Beijing 100083, China

⁴ Key Laboratory of Agricultural Land Quality and Monitoring, Ministry of Natural Resources, Beijing 100083, China

⁵ State Key Laboratory of Earth Surface Processes and Resource Ecology, Beijing Normal University, Beijing 100875, China; yesj@bnu.edu.cn

* Correspondence: yxc@cau.edu.cn; Tel.: +86-188-1135-6282

Received: 1 September 2020; Accepted: 22 October 2020; Published: 25 October 2020



Abstract: Recently, increasing amounts of multi-source geospatial data (raster data of satellites and textual data of meteorological stations) have been generated, which can play a cooperative and important role in many research works. Efficiently storing, organizing and managing these data is essential for their subsequent application. HBase, as a distributed storage database, is increasingly popular for the storage of unstructured data. The design of the row key of HBase is crucial to improving its efficiency, but large numbers of researchers in the geospatial area do not conduct much research on this topic. According the HBase Official Reference Guide, row keys should be kept as short as is reasonable while remaining useful for the required data access. In this paper, we propose a new row key encoding method instead of conventional stereotypes. We adopted an existing hierarchical spatio-temporal grid framework as the row key of the HBase to manage these geospatial data, with the difference that we utilized the obscure but short American Standard Code for Information Interchange (ASCII) to achieve the structure of the grid rather than the original grid code, which can be easily understood by humans but is very long. In order to demonstrate the advantage of the proposed method, we stored the daily meteorological data of 831 meteorological stations in China from 1985 to 2019 in HBase; the experimental result showed that the proposed method can not only maintain an equivalent query speed but can shorten the row key and save storage resources by 20.69% compared with the original grid codes. Meanwhile, we also utilized GF-1 imagery to test whether these improved row keys could support the storage and querying of raster data. We downloaded and stored a part of the GF-1 imagery in Henan province, China from 2017 to 2018; the total data volume reached about 500 GB. Then, we succeeded in calculating the daily normalized difference vegetation index (NDVI) value in Henan province from 2017 to 2018 within 54 min. Therefore, the experiment demonstrated that the improved row keys can also be applied to store raster data when using HBase.

Keywords: geospatial big data; HBase; row keys; large scale; storage; GF-1 imagery

1. Introduction

The volume of the multi-source geospatial data from Earth observation systems, such as satellites, meteorological stations and so on, is currently increasing quickly worldwide. Earth observation systems have generated large amounts of data, which can reach to the range of petabytes; in the future, the volume will probably reach the exabyte level or even greater than that [1]. The data have numerous potential values, but we are usually able to only utilize a small part of them in a research domain after filtering from large numbers of datasets; thus, it is necessary to store all observed data, as we cannot predict which data we can utilize in a research before filtering. That is, large amounts of storage resources are required to store multi-source geospatial data generated in short intervals. Therefore, saving storage resources is an important issue globally. These data are not only large in volume but also of various formats, such as raster data [2], textual data [3], vector data [4], etc.; therefore, storing and organizing these data efficiently is essential for the subsequent applications of multi-source geospatial data, such as data fusion [5–7], data assimilation [8–10] and so on.

To date, large numbers of research works have been conducted that have focused on using different kinds of strategies or regulations to manage and organize these geospatial data. It seems that most researchers have reached a consensus to some degree regarding the use of geospatial grids [11–14]. Theoretically, regarding different research scales, a grid can be divided into two categories: one category is the discrete global grid, and the other is the local planar projection-based grid [15]. The discrete global grid contains a regular polyhedron-based grid system [16–19], a sphere VORONOI-based grid system [20,21] and a longitude/latitude line-based grid system [22–24]. This kind of grid can cover the entire globe and has hierarchical and recursive characteristics, but it generally has an extremely complicated computation process [25,26]. A local planar projection-based grid generally utilizes a hierarchical square kilometer grid to manage geospatial data [13]. Although this kind of grid has some distortions after being projected to a plane from a sphere, it can simplify the computing process. Moreover, if the process of projection is completed by splitting the entire globe into zones (e.g., split by longitudes) in advance and then projecting each zone to the plane, it can reduce this distortion. This is why the Universal Transverse Mercator (UTM) projection consists of 3° UTM and 6° UTM. In this work, we selected a kind of planar projection-based grid to achieve the logical management of geospatial data, which will be illustrated in Section 2.1.

There also are a great number of researchers who are paying attention to solutions regarding the storage of such massive data; among these, a distributed file system, such as the Hadoop Distributed File System (HDFS), is a possible option [27–31]. HDFS is a sub-core program of the Hadoop program from the Apache Software Foundation, which is a non-profit organization that aims to support open source software programs. However, HDFS is designed for large datasets and will incur large stresses for the master node if we store massive small files, because the master node needs to store a metadata for each small file stored in the slave node [32]. In order to solve this problem, HBase, which is a database based on HDFS, was developed [33]. The database is famous for its flexibility for storing large amounts of unstructured data and the ability to expand to unlimited columns and rows, which can solve the problem of storing massive small files [34]. In this work, as mentioned above, we selected a grid strategy to organize data, which means that our geospatial data that need to be stored are presented as a large number of small files. Therefore, we decided to use HBase to store our geospatial data, in the same manner as some other researchers. Regarding the efficient use of HBase, the design of the row keys is one of the essential issues. HBase stores data as a pattern of key values, which means that there is always a row key that needs to be stored together with a value that we want to store [35]. Thus, row keys are meant to be kept as short as is reasonable while still being useful for the required data access, which can save large amounts of storage resources and improve the efficiency, but most researchers do not pay attention to this problem. It is worth noting that a short key that is useless for data access is not better than a longer key with better get/scan properties; we should expect tradeoffs when designing row keys [36]. In fact, most researchers are simply interested in designing their row

keys to include the information that they think they need and rarely notice the problem of the excessive length of row keys [14,37–40].

In this work, we proposed a method to solve this problem. We used ASCII codes to substitute the original geospatial codes, which can shorten the length of the row keys of the HBase. We also designed experiments to test the effect of saving storage resources and compared the time consumption of queries. The proposed method can provide a new concept for the design of row keys for all researchers when they intend to utilize HBase to store their data.

This paper is organized as follows. Section 2 introduces the spatial grid strategy we selected, the original row key obtained by the spatial grid strategy and our improved method to shorten the original row key. Section 3 demonstrates the effect of the proposed method. Finally, Sections 4 and 5 discuss the experimental results and list future work.

2. Methodology

Based on our previous research and knowledge, we decided to select and utilize the Raster Dataset Clean and Reconstitution Multi-Grid (RDCRMG) [13] as the spatial index grid to clip, store and organize our multi-source data. In this section, in order to make it easier for readers to understand the design of the row keys later in the paper, the partition and coding strategy of the RDCRMG is presented. According to the structure of the RDCRMG, the original spatio-temporal design of the table in the HBase is explained; then, a more functional table structure is elaborated to address the shortcomings of the original table. Meanwhile, the improved spatio-temporal design method of the row keys based on the ASCII codes is also proposed.

2.1. Spatial Index Multi-Grid

2.1.1. Spatial Reference

The RDCRMG spatial reference is the World Geodetic System 1984 (WGS 84)-based Universal Transverse Mercator (UTM) 6° strip division projection coordinate system, which has the following peculiarities: firstly, there is a significant capability to improve the management efficiency because of the explicit spatial mathematical foundation, partitioning rule and conversion algorithm for the grid code and spatial coordinates; secondly, the system can maintain consistency when subdividing the spatial data into extent grids, and it is impossible for data to belong to one grid and other grids; thirdly, compared with other projections (e.g., the conical or azimuthal projection, the Gauss–Kruger projection), higher accuracy can be maintained and less distortion on the boundary of the projection zone; lastly, the system is helpful when conducting data-intensive calculation, as it provides the possibility to compute in parallel according to its small grids instead of computing the entirety of the data.

2.1.2. Partition and Coding

The RDCRMG splits the entire geographic area (e.g., China) into several zones with 6° longitude. In the each zone, the RDCRMG contains a hierarchical grid strategy that is composed of 100 km grids and 10 km grids (there is another layer in the RDCRMG—the 1 km grid—but in order to highlight the research in this paper, we ignored the 1 km grid to make the structure more simple). These two levels of square grids are generated with strict nested relationships, as shown in Figure 1. The grids in the same level have the uniform size, shape and orientation, and there is no seam between two adjacent grids. Therefore, while storing geographic data, the data should be split or cropped into small blocks according to the boundary of the grids which are overlapped with these data. Furthermore, the RDCRMG adopts the row–column structure rather than the quad-tree structure, because the RDCRMG focuses on data extraction efficiency, lower query algorithm complexity and higher organizational pattern consistency.

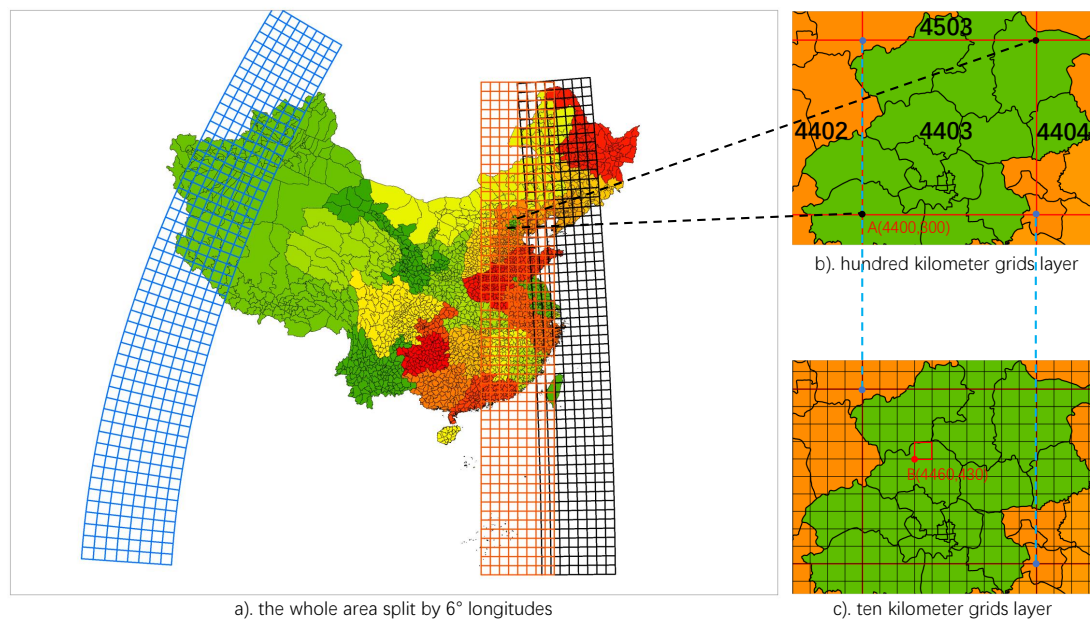


Figure 1. Raster Dataset Clean and Reconstitution Multi-Grid (RDCRMG) grid partition. (a) The whole area split by 6° longitudes. (b) The 100 km grid layer. (c) The 10 km grid layer.

In terms of grid coding, the 100 km grid code is composed of four digits: the first two digits refer to the y-coordinate of the grid's southwest vertex (unit: km), and the last two digits represent the x-coordinate. When using a code to calculate the coordinates of a 100 km grid, we only need to multiply the two parts by 100 km. For example, as Figure 1b shows the y-coordinate and x-coordinate of the 100 km grid's southwest vertex (A) are 4400 (km) and 300 (km), respectively. Therefore, the grid's code is 4403. Due to the limit of the distortion and spatial scale, the maximum of the 100 km grid's first two digits is 59, and the range of the 100 km grid's last two digits is from 00 to 09. As regards the 10 km grid, two additional digits are used to represent the position. The two digits' incremental direction is consistent with a z curve from southwest to northeast within a 100 km grid (from 00 to 99). As shown in Figure 1c, the y-coordinate and x-coordinate of the southwest vertex (B) are 4460 (km) and 430 (km), respectively, and the code of the corresponding 10 km grid (the red square in the figure) is 440363.

Moreover, the RDCRMG stores the data by the directory name and file name to generate a logical storage path of each data block without any metadata, as shown in Figure 2. Root directories are named after the spatial coordinate system WKID (the spatial reference system's well-known ID) and correspond to different UTM projection strips. Then, other subdirectories are named after the 100 km code, 10 km code and year. Eventually, the file blocks are stored in directories by year. The file block also has its own specific name codes, as shown in Figure 3. Data type codes are used to distinguish different data, such as GF1WV (001), Sentinel 2 (002) and crop classification (005). A random code is used to avoid overriding files with the same name in the form of letters or figures.

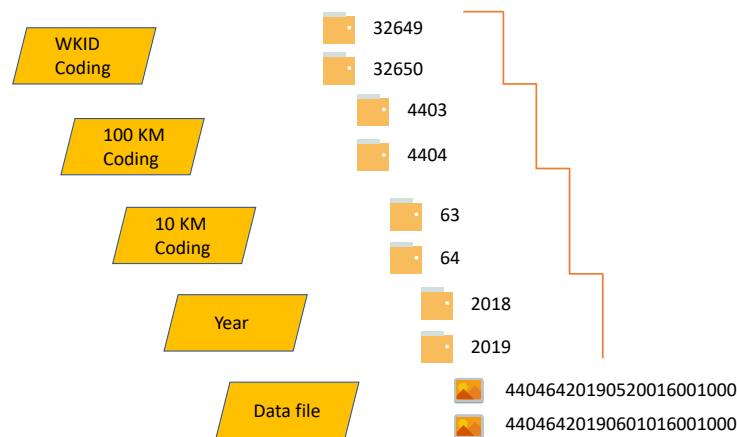


Figure 2. RDCRMG-based file block storage path.

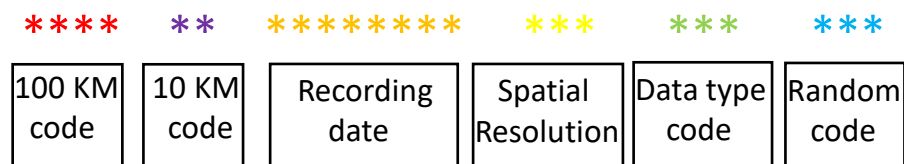


Figure 3. RDCRMG-based file block name code structure.

2.2. Spatio-Temporal HBase Table Based on RDCRMG

2.2.1. The Structure of a HBase Table

Before introducing our spatio-temporal HBase table, we present the structure of a general HBase table from the logical perspective and physical perspective. The logical structure is shown in Table 1. It contains the row key, column family, column, timestamps and value. Moreover, timestamps (t1, t2, t3, t4) are used to rank data values; if we keep the default set, this will be the exact time at which we save the data.

Table 1. The logical structure of the HBase table.

RowKey	Column-Family-A		Column-Family-B			Column-Family-C
	Column-A	Column-B	Column-A	Column-B	Column-C	Column-A
Key001	t2: xx t1: yy			t4: xx t3: yy t2: nnn		
Key002		t3: xx			t4: xx t1: yy	
Key003	t3: xxx t1: yy		t4: xx t3: yy t2: nnn			t2: kk t1: yy

The physical structure is shown in Figure 4. For an HTable, each HRegion server manages some HRegions. HRegion contains the HStore, and the number of the HStore depends on the number of the column family. Then, each HStore is composed of MemStore and StoreFile. With an increasing data volume, a large HRegion has to be split into two small HRegions to meet the restrictions of the data volume for each HStore.

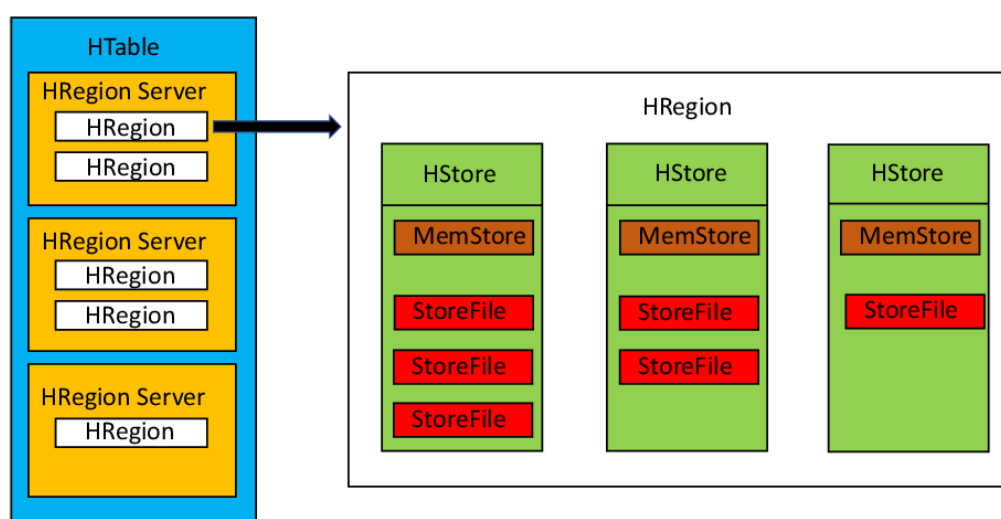


Figure 4. The physical structure of the HBase table.

2.2.2. The Design of the Row Key

The design of the row key is pivotal for the HBase table structure. HBase's first-level index is its row key, which means that, with regard to the spatio-temporal data, it would be better to store spatio-temporal information in the row key rather than the column family or columns. Otherwise, it would take more time for HBase to query data with the entered spatio-temporal conditions [36]. Therefore, we deliberately put spatio-temporal information regarding the data into the row keys. As mentioned above, the RDCRMG has three scale layers: 100 km, 10 km and 1 km. In this paper, we only discuss the HBase design for the 10 km spatio-temporal grids; similarly, we can achieve the HBase table for other scale grids with the same strategy.

From Figures 2 and 3, we can gather the information we need for the design of the row keys. The location information is the WKID code, the 100 km code and the 10 km code, and the time information is the recording date. Considering that our application is more likely to query time series data, we decided to combine that information together, as shown in Figure 5.



Figure 5. The structure of the original row key.

The combination of the WKID, 100 km code and 10 km code is able to ensure a unique space; meanwhile, the recording date limits the sole time. Thus, a row key represents a unique time and space. Moreover, the location information is in front of the time information, which guarantees that the time series data of a geospatial grid is stored in one physical block or adjacent blocks. In this way, region servers can more simply find the related spatio-temporal data of a 10 km grid. In order to make citation more convenient in the next sections, we refer to this kind of row key as the original row key, and this method is termed the original row key encoding method.

2.2.3. The Design of the Column Family

Theoretically, an HBase table can contain more than one column family, and each column family can have a large number of columns. However, HBase will create an HStore for each column family; that is, if we build up a few column families, when HBase has to conduct the split operation to reduce the volume of some data regions, all HStores must also be split. In a real application, we could not control the data volume for each column family, meaning that some HStores would increase rapidly to cause the HBase to split the regions, but some HStores which were small would also be split into numerous new HStores, and thus the HRegion server would have to manage more HStores. Besides the split operation, the flush operation from the MemStore would also lead to more I/O consumption because of the increased number of column families. Therefore, in this study, we only adopt one column family with a large number of columns to save data.

2.3. Improved Spatio-Temporal Model

The main idea of the improved spatio-temporal model is to use the ASCII code to substitute the original code.

According to the official instructions of HBase [36], it would be better to shorten the lengths of the row keys and other qualifiers along with the names of the column families, columns and so on. Therefore, if we could shorten the lengths of these labels, we might save more storage resources and speed up data retrieval. The ASCII code table is shown in Table 2. In fact, in our approach, we would not use all ASCII codes, because some characters are hard to print manually with our keyboard. Thus, we only use the chars from " " (space) to "~"; the corresponding decimals are from 32 to 126.

Table 2. The American Standard Code for Information Interchange (ASCII) Code Table (Dec represents decimal number).

Dec	Character	Dec	Character	Dec	Character	Dec	Character
0	NUL (null character)	32	space	64	@	96	'
1	SOH (start of header)	33	!	65	A	97	a
2	STX (start of text)	34	"	66	B	98	b
3	ETX (end of text)	35	#	67	C	99	c
4	EOT (end of transmission)	36	\$	68	D	100	d
5	ENQ (enquiry)	37	%	69	E	101	e
6	ACK (acknowledge)	38	&	70	F	102	f
7	BEL (bell (ring))	39	'	71	G	103	g
8	BS (backspace)	40	(72	H	104	h
9	HT (horizontal tab)	41)	73	I	105	i
10	LF (line feed)	42	*	74	J	106	j
11	VT (vertical tab)	43	+	75	K	107	k
12	FF (form feed)	44	,	76	L	108	l
13	CR (carriage return)	45	-	77	M	109	m
14	SO (shift out)	46	.	78	N	110	n
15	SI (shift in)	47	/	79	O	111	o
16	DLE (data link escape)	48	0	80	P	112	p
17	DC1 (device control 1)	49	1	81	Q	113	q
18	DC2 (device control 2)	50	2	82	R	114	r
19	DC3 (device control 3)	51	3	83	S	115	s
20	DC4 (device control 4)	52	4	84	T	116	t
21	NAK (negative acknowledge)	53	5	85	U	117	u

Table 2. Cont.

Dec	Character	Dec	Character	Dec	Character	Dec	Character
22	SYN (synchronize)	54	6	86	V	118	v
23	ETB (end transmission block)	55	7	87	W	119	w
24	CAN (cancel)	56	8	88	X	120	x
25	EM (end of medium)	57	9	89	Y	121	y
26	SUB (substitute)	58	:	90	Z	122	z
27	ESC (escape)	59	;	91	[123	{
28	FS (file separator)	60	<	92	\	124	
29	GS (group separator)	61	=	93]	125	}
30	RS (record separator)	62	>	94	^	126	~
31	US (unit separator)	63	?	95	_	127	DEL

2.3.1. Row Key Based on ASCII Code

The structure of the proposed row key is shown in Figure 6. In this part, we refer the reader to Section 2.1.2 and Figure 5 for an explanation of the transformation of the original code and the ASCII code.

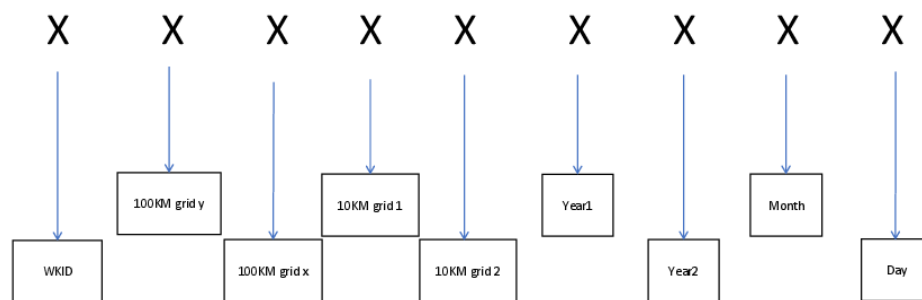


Figure 6. The structure of the proposed row key.

The first symbol represents WKID. The Universal Transverse Mercator (UTM) projection is coded every 6° from west to east; the first zone number is 32,601 and the last zone number is 32,660, and so we use an ASCII code to represent these zones instead of five numbers. The approach used for the transformation is to use the last two numbers of each zone and to add 32; then, the corresponding ASCII code is obtained. For the 32601 zone, we use $01 + 32$ to get 33 and then transform the number to the ASCII code, "!". Therefore, we could use the symbol ("!") to substitute 32,601. Moreover, the reason for adding 32 is that it is difficult to print the first 32 symbols of the ASCII code directly—particularly as we would sometimes need to operate data manually—and thus we decided to add 32 to make it easier to print the symbol.

The second symbol represents the first two digits of the 100 km grid (100 km grid y), and the third symbol is the last two digits of the 100 km grid (100 km grid x). For 100 km grid y, the original code is 00 to 59, which means from equator to 60° N, and so we add up the numbers and 32 to obtain an ASCII code with a range from " " (space) to "[". As regards the 100 km grid x, we utilize "1" to "8" instead of the original code, which is from 01 to 08.

The fourth and fifth symbol are the 10 km code. The range of each symbol is from "0" to "9". There is no change compared to the original code.

We split the information regarding the year into Year1 (the first three digits of the year) and Year2 (the last digit of the year), as the sixth and seventh symbols show. The value range of Year2 is obviously from 0 to 9, and so we only need to use the same ASCII char (from "0" to "9") to present it. Regarding Year1, we run into difficulty because it always over 190 and obviously exceeds the range of the ASCII Code Table. Thus, we decide to use a new method to substitute this value. For example, 190 is divided into 19 and 0, and then 19 minus 15 is performed to get 4; next, we combine 4 with 0 to get 40, and the

corresponding ASCII char is "(" . Basically, this method has its own positive and negative aspects; the advantage is that we successfully reduce the number of digits from 3 to 2, meaning that the code is able to be found in the ASCII Code Table. The restriction is that we can only use this method to deal with numbers from 182 to 276, meaning that we can only store data produced from 1820 to 2769. As regards the month and day, we add them and add 31, respectively, to get the ASCII code.

2.3.2. Columns Based on ASCII Code

Changing the length of the row key is not the only way to shorten the lengths of the key–value pairs. From Figure 7, we can see that the key–value pairs consist of three parts: length information, key and value. In the length information part, both KeyLength and ValueLength are constants; they occupy 4 bytes, respectively. In the key and value parts, the lengths of some information are variable, which means that, if we want to shorten the lengths of the key–value pairs, reducing the lengths of these parameters is essential. The row key mentioned above is one of these parameters, but not the only one. We should also pay attention to the lengths of the family and the qualifier (the column name). We have discussed the fact that we only use one column family in Section 2.2.3; thus, we utilize one ASCII char "T" to name our Column Family. "T" is the acronym of the word "type"—which means that we would store different types of data in this column family.

4	4	2	variable	1	variable	variable	8	1	variable
KeyLength	ValueLength	RowKey Length	RowKey	Family Length	Family	Qualifier	Timestamp	Type	Value
Key								Value	

Figure 7. The structure of the key–value pairs.

With respect to naming the columns, there are some differences between meteorological station data and remote sensing imagery. For meteorological station data, we could create a column named "M" to store the station ID, longitude and latitude, or other information about this station. We could acquire 12 daily meteorological indicators from stations, including the average atmospheric pressure, average temperature, rainfall and another 21 indicators. Therefore, according to the useful range of the ASCII code mentioned above, we decide to combine "M" and another character to name each indicator's column; this character ranges from "!" to "8" in the ASCII Code Table. That is, the name of the column used to store the average atmospheric pressure is "M!"; the name of the column used to store the average temperature is "M""; and so on; the last column's name is "M8".

For remote sensing imagery, we decided to use four characters (ABCD) to store metadata and three characters (EFG) to store imagery. Regarding "ABCD," we put the character "a" into the first place (A), meaning that this column is for metadata. The character in the second place (B) begins from ""—meaning where the imageries come from; for example, " " represents GF1WV, "!" represents Sentinel2, "" represents Landsat8 and so on. The third place (C) is designed for the cloud percentage of each cropped image, and the range is from "0" to "9"; for instance, "0" means the cloud percentage is [0%, 10%), "1" shows ghat the cloud percentage is [10%, 20%) and so on. The last place (D) has two values: "A" and "B." "A" means that the name of this cropped image is stored, while "B" means that the original imagery from which this cropped image came from is recorded. Regarding "EFG," the character in the first place (E) is "b"—meaning that this column is for image data. The meanings of F and G are the same as B and C, respectively. For example, if we crop an original GF1WV image (e.g., the name is GF1_WV1_E78.2_N39.6_20180816_L1A0003394802) into large numbers of small, cropped images, one of these small cropped image's names is 44036320180816016001000, and its cloud percentage is 25%. When we store this small cropped image, "GF1_WV1_E78.2_N39.6_20180816_L1A0003394802" would be stored in the column named "a 2B"; "44036320180816016001000" would be stored in the column named "a 2A"; and the real image named 44036320180816016001000.tif would be stored in the column

named "b 2." For example, if we acquire new meteorological data and GF1WV remote sensing data of spatio-temporal grid A (the spatial grid is 32650440363, the date is 20180816), these data would be stored in HBase as shown in Figure 8.


	T							
	M	M!	M8	a 2A	a 2B	b 2
RL36338/	Information of meteorological station	Average atmosphere pressure	Sunshine Duration	4403632018 0816016001 000	GF1\WV 1\E78.2\ N39.6\20 180816\L 1A0003394 802	 440363201 808160160 01000.tif

Figure 8. How the data of code 3265044036320180816 would be stored in the HBase Table.

3. Results

3.1. Experiment Design

In this paper, we designed three experiments to demonstrate that the proposed row key encoding method is efficient for meteorological station textual data and remote sensing imagery. The first experiment made a comparison between the original row key encoding method and the proposed row key encoding method to demonstrate whether the proposed method could save storage resources. The second experiment compared the data query efficiency of these two methods. The last experiment involved a simple application which produced each GF-1 image's NDVI layer in Henan Province, China from 2017 to 2018; this experiment could preliminarily demonstrate that the proposed row key encoding method is able to be used for the subsequent spatio-temporal calculation.

3.2. Row Key Compression Efficiency

In order to explore the advantages and disadvantages of the proposed row key encoding method in compression, we used two different kinds of data: meteorological station textual data and remote sensing imagery.

For meteorological data, we stored daily meteorological indicators of 831 meteorological stations in China from 1985 to 2019. Then, we adopted four patterns to make a comparison, including the key–value volume based on the original row key encoding method, the key–value volume based on the proposed row key encoding method, the key volume based on the original row key encoding method and the key volume based on the proposed row key encoding method; the result is shown in Figure 9. Obviously, four lines in the figure almost increase linearly with the increase of the rows; meanwhile, the proposed row key encoding method could reduce resource consumption irrespective of the key volume or the key–value volume compared with the original method. For the whole meteorological data (7,946,627 rows) stored in the HBase, we could save 1874 MB of storage resources, and the compression percentage of the key was 29.41%, while the compression percentage of the row key was 52.63%. The compression percentages of the key and row key were relatively stable, but the compression percentage of the key–value pairs had a strong relation with the volume of data stored in each column of the HBase table. For this experiment, the compression percentage of the key–value pairs was 20.69%, which demonstrates that the proposed method is able to save more storage resources when we store meteorological station textual data in HBase.

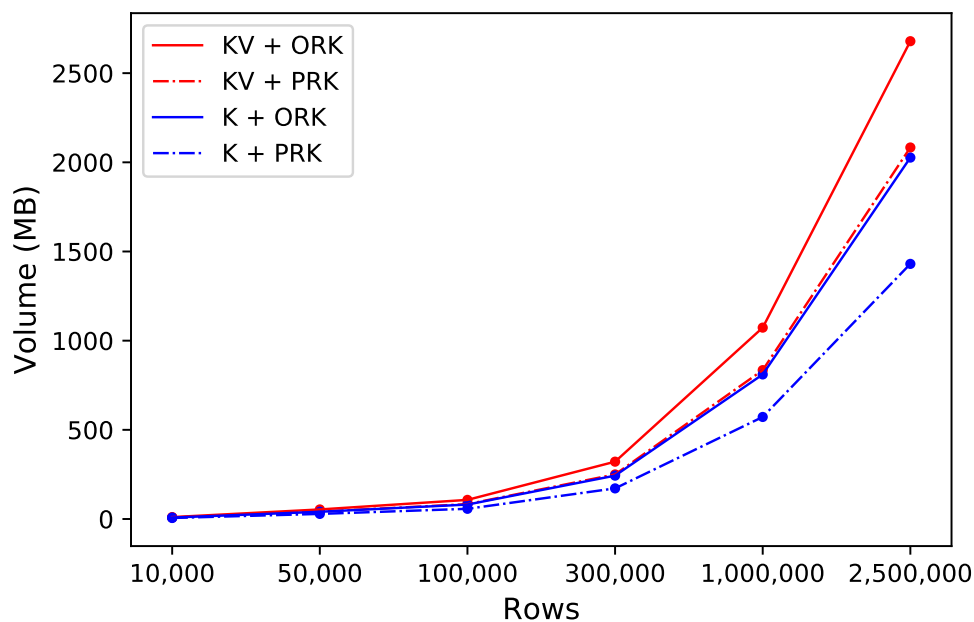


Figure 9. The compression efficiency with meteorological station textual data (KV: key–value; K: key; ORK: the original row key; PRK: the proposed row key).

For remote sensing imagery, we calculated the volumes of the value and the row key of different rows, as Figure 10 shows.

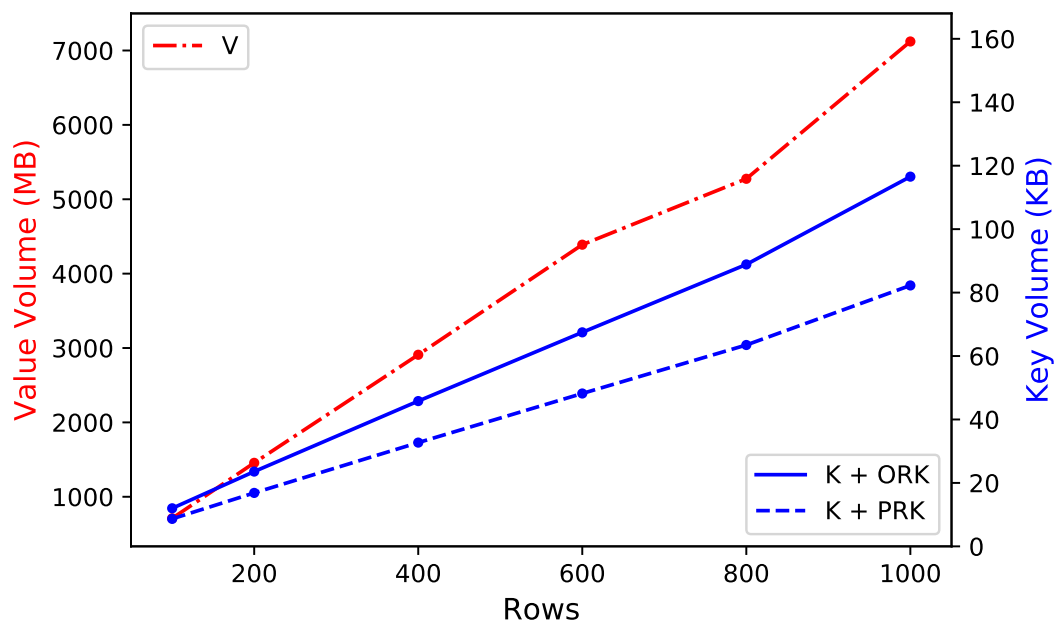


Figure 10. The compression efficiency with remote sensing imagery (V: value; K: key; ORK: the original row key; PRK: the proposed row key).

We can see that the proposed method hardly affects the key–value pairs' volume, because the volume of remote sensing imagery is far larger than the volume of the key or the row key (the unit of

the key volume is kilobytes (KB), the unit of the value volume is megabytes (MB)). In fact, the proposed method affects the length of the row key or the key and thus affects all key–value pairs, so if there is a large volume difference between the value and the key, this method would not have satisfactory efficiency in terms of compressing the key–value pairs’ volume. However, for the compression percentage of the key, the mean is 28.57%, which means that, if there are numerous rows and columns, it would also save large amounts of storage resources; of course, this is limited to the comparison of the storage resources that the remote sensing imagery needs to occupy. Moreover, the speed of the change of volume as the rows increase is somewhat different to that in Figure 10 but nearly the same as in Figure 9. That is because the amount of meteorological station textual data that needs to be stored each day for each meteorological station is a constant, so the HBase table would have a stable increase of rows and columns. However, the amount of the remote sensing imagery which needs to be stored is variable for each row; therefore, the HBase table would have a stable increase for rows and an unstable increase for columns. For remote sensing imagery in this experiment, the range of the columns for each row is from three to nine.

3.3. Data Query Efficiency

In this experiment, we explore the query efficiency of the proposed row key encoding method and the original method with two query types: one is a random query and the other is a region query. The experimental data are daily meteorological station textual data from 831 meteorological stations in China from 1985 to 2019.

3.3.1. Random Query

In the same computer cluster, we randomly selected 562 rows of data and 1126 rows of data and then calculated the time consumption of queries for different rows of data based on these two different row key encoding methods; the result is shown in Table 3. From the result, we can see that the time consumptions of the random queries for these two methods are almost equivalent. However, their time consumptions are far greater than the query efficiency of some relational databases; for example, MySQL. This kind of spatio-temporal index is therefore not a good choice for a random query.

Table 3. The efficiency of the random query for different methods (ORK: the original row key; PRK: the proposed row key).

Rows	Method	Time1 (ms)	Time2 (ms)	Time3 (ms)	Time4 (ms)	Time5 (ms)	Mean (ms)
562	ORK	90,837	89,452	89,775	90,451	88,451	89,793
	PRK	87,978	89,799	89,951	88,121	90,453	89,260
1126	ORK	183,206	191,340	187,665	184,562	186,785	186,711
	PRK	189,969	187,543	184,568	185,465	185,461	186,601

3.3.2. Region Query

According to the design of the row key, any 10 km grid’s temporal data are supposed to be stored in the same data region or some adjacent data regions. Therefore, for the different row key encoding methods, we randomly selected a 10 km grid to calculate the time consumption of the query with different time lengths. The result is shown in Figure 11. We also considered the time consumption of the decoding for the proposed row key encoding method in the result. Through the lines corresponding to the mean time in the figure, we can see that, although when using the proposed method, the row key has to decode the obscure ASCII code, the proposed method still needs less time for the query compared with the original method. If we query a longer date, we can save more time by using the proposed method, but for each query, there are some variations in terms of time consumption, as shown by the blue and red rectangles in the figure. When we analyze the time consumptions of the region query and the random query, we can see that this kind of spatio-temporal index is not efficient

for a random query but is useful for a region query. The last data period, 19850101–20130418, contains around 9700 rows of data and the mean time consumption is about 1.39 s, which is again faster than the random query.

According to the design of RDCRMG, the code for the spatial grid is before the time series code, which means that we choose to store the time series data of a grid continuously rather than the neighboring grid. Therefore, this kind of efficient region query would work on time series but not neighboring grids in the spatial dimension. If we want to obtain the best efficiency for the spatial query of a neighboring grid, we should invert the code order of the space and time series, which could bring the neighboring grid of a certain date into the same region.

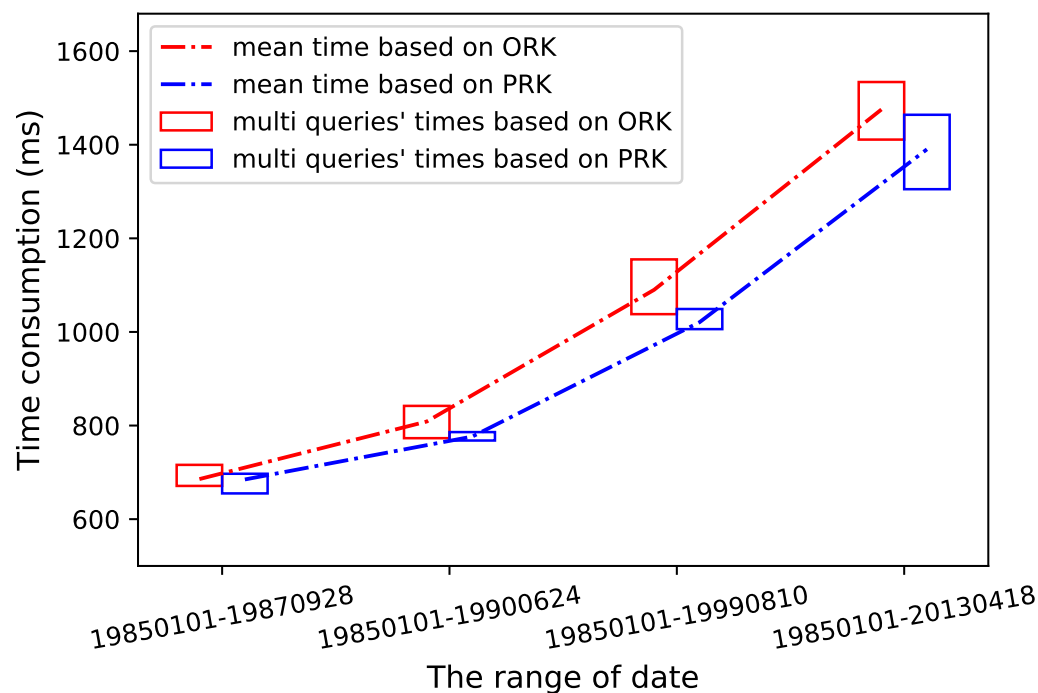


Figure 11. The efficiency of the region query with different methods (ORK: the original row key; PRK: the proposed row key).

3.4. Application to Spatio-Temporal Calculation

In this experiment, we stored some of the GF-1 imagery in Henan Province, China from 2017 to 2018 into HBase with the proposed row key encoding method. We set some parameters, including the spatial range (Henan Province), the time period (from 2017 to 2018), the data type (the GF-1 imagery), the calculation model (NDVI) and the cloud percentage of imagery (we used two cloud proportions: filter1: 0–100% and filter2: 0–50%). Then, we adopted the Map-Reduce paralleled calculation mode to determine the NDVI layers in time series in Henan Province with different cloud percentage conditions; the result is shown in Figure 12. Moreover, when we set a cloud percentage filter—for example, 0–50%—that meant that HBase would try to provide the imagery with the lowest cloud percentage within 0% to 50% per day and per spatial 10 km grid. The purpose of this design is to provide the imagery with less cloud contamination for large amounts of subsequent spatio-temporal calculation as far as possible. That is also the reason that the column names for remote sensing imagery were designed, as mentioned in Section 2.3.2. HBase would scan a row's columns from left to right; therefore, for example, the column named "b 2" (where the cloud percentage was (20%, 30%)) would be sorted before the column named "b 4" (where the cloud percentage was (40%, 50%)). Thus, HBase is able to find the column named "b 2" faster than the column named "b 4"—which is exactly the desired outcome. Of course, if clouds are the object of research, this design should be inverted. From Figure 12,

we can see that, even though we set no restriction on cloud percentage, there were only 67 days which had NDVI layers. This is because the GF-1 remote sensing satellite's visit period is four days, and we downloaded most of the images but not all of them. Furthermore, we can see that if we set the cloud percentage to less than 50%, there are more days with a lack of a corresponding NDVI layer, which is reasonable. The time required for both of these calculations is around 54 min. This experiment could preliminarily demonstrate that the designed spatio-temporal index is effective for subsequent spatio-temporal calculation.

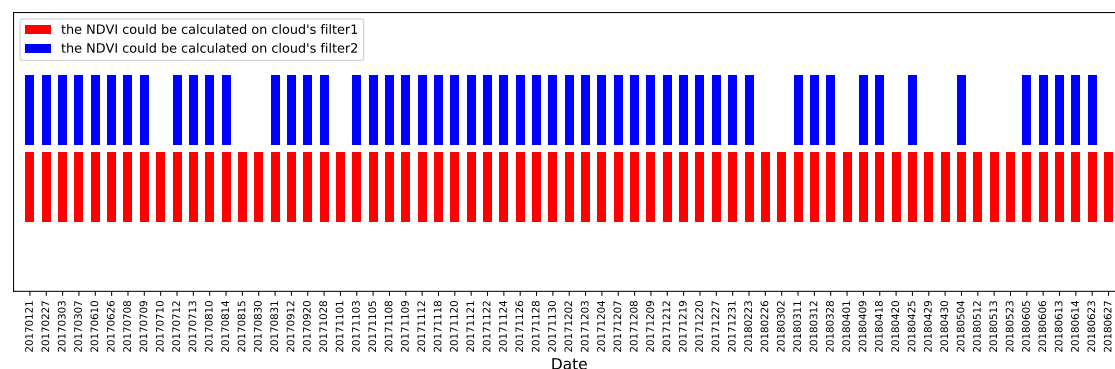


Figure 12. The time distribution of the normalized difference vegetation index (NDVI) results based on different filters of cloud percentage (filter1: the cloud percentage is between 0 and 100%; filter2: the cloud percentage is between 0 and 50%).

4. Discussion

The proposed row key encoding method could shorten the length of the row key, which is critical to saving more storage resources for HBase, which needs to repeatedly store the row key for each column. The more columns each row has, the more storage resources would be saved. In this paper, our data are large-scale spatio-temporal data, which means that when we put the data into a real application, the table must have a large number of columns as there are large amounts of multi-source spatio-temporal data. Therefore, this method would have its own prospective application. However, its efficiency depends on how much information the original row key has and how long the length of the original row key is. If the original row key is extremely simple and there is no way to use a short string to substitute a long string for some information, this method would not have a satisfactory effect.

From the results, we could see that the efficiency of the random query was unsatisfactory but the efficiency of the region query was acceptable. This has nothing to do with the proposed row key encoding method but is related to the original spatio-temporal index. There is no design which could fit every application; for our subsequent applications, we would need to fetch spatio-temporal data for long time series, and so the efficiency of the region query was more important to us. If it desired that the efficiency of the random query should be improved, a hash function should be used with the addition of "salt" for the original index, which would allocate data to each data node in a disordered manner. However, this kind of method usually has a negative effect on the region query (although one could also build some second level index tables (auxiliary index tables)).

We also briefly explored other methods to see whether they would save more storage resources or improve the efficiency of the query. One of the typical methods is "prefixtree," which is a kind of key encoding method. A prefix tree is also known as a Trie; it is used to optimize search complexities. We implemented four groups of experiments: the original row key encoding method and non-prefixtree, the proposed row key encoding method and non-prefixtree, the original row key encoding method and prefixtree and the proposed row key encoding method and prefixtree. We found that the prefixtree method would have a greater apparent efficiency for the experimental data used in this paper compared with the proposed row key encoding method. However, the test showed that the result of the combination of these two methods was best in terms of the efficiency of saving storage and the

efficiency of the query. The reason for this is that these two methods are not contradictory; they optimize the key of HBase in different dimensions. The "snappy" method could also save storage resources, but this method always acts on the value of the HBase table (especially when the values are large and not precompressed) rather than the row key or the key, and so this method exceeds the scope of research in this paper. In the future, we aim to pay attention to how to compress the value stored in the HBase's columns.

There is another issue for the spatio-temporal index used in the paper. We attempted to store the adjacent data in terms of space and time in the same data region or the adjacent data region, which is useful for a region query, but it also caused a hotspot issue for storage and query. We also attempted to utilize the presplit policy and automatically split policy to solve the hotspot issue for storage, which seemed to be preliminarily effective. We would like to address this problem in the future.

5. Conclusions

In order to save more storage resources and improve the speed of query for HBase, we proposed a method with shorter ASCII characters to shorten the length of the original row key created by the Raster Dataset Clean and Reconstitution Multi-Grid (RDCRMG). The results show that our method could not only save storage resources when it comes to the key of the HBase (with a compression ratio of 29.41%), but also that it could have excellent efficiency for a region query compared with the original row key. This method changes the conventional thought behind the design of the row key policy. For other applications, researchers are also able to use shorter ASCII characters to substitute longer information according to the method proposed in this paper. Moreover, when long keys (compared to the values) or many columns are used, we could simultaneously use the prefixtree method and the proposed method to reduce the key's data volume and improve the speed of the region query. We also used the map-reduce paralleled calculation mode to fetch spatio-temporal data from the HBase and accomplished the NDVI calculation for Henan Province from 2018 to 2019. Thus, we could preliminarily demonstrate that our designed spatio-temporal storage model is effective for subsequent spatio-temporal application. Based on this, we will be able to integrate more spatio-temporal calculation models into our research.

Author Contributions: Conceptualization, Quan Xiong; methodology, Quan Xiong; software, Wei Liu, Zhenbo Du and Quan Xiong; validation, Sijing Ye, Quan Xiong and Wei Liu; formal analysis, Zhe Liu; investigation, Quan Xiong; resources, Sijing Ye and Zhenbo Du; data curation, Diyou Liu; Writing—original draft preparation, Quan Xiong; Writing—review and editing, Quan Xiong and Xiaochuang Yao; visualization, Wei Liu; supervision, Xiaodong Zhang; project administration, Dehai Zhu; funding acquisition, Xiaochuang Yao. All authors have read and agreed to the published version of the manuscript.

Funding: This research is supported by the National Key Research and Development Program of China (project No. 2018YFE0122700) and the National Earth Observation Data Center.

Acknowledgments: The paper is funded by the China Scholarship Council.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ASCII	American Standard Code for Information Interchange
GF-1	GaoFen No.1
NDVI	Normalized Difference Vegetation Index
UTM	Universal Transverse Mercator
HDFS	Hadoop Distributed File System
RDCRMG	Raster Dataset Clean and Reconstitution Multi-Grid
WGS 84	World Geodetic System 1984
WKID	Spatial Reference System's Well-Known ID

References

1. Nativi, S.; Mazzetti, P.; Santoro, M.; Papeschi, F.; Craglia, M.; Ochiai, O. Big data challenges in building the global earth observation system of systems. *Environ. Model. Softw.* **2015**, *68*, 1–26. [\[CrossRef\]](#)
2. Zhu, X.; Cai, F.; Tian, J.; Williams, T. Spatiotemporal fusion of multisource remote sensing data: Literature survey, taxonomy, principles, applications, and future directions. *Remote Sens.* **2018**, *10*, 527.
3. Wei, X.; Duan, Y.; Liu, Y.; Jin, S.; Sun, C. Onshore-offshore wind energy resource evaluation based on synergetic use of multiple satellite data and meteorological stations in Jiangsu Province, China. *Front. Earth Sci.* **2019**, *13*, 132–150. [\[CrossRef\]](#)
4. Yao, X.; Li, G. Big spatial vector data management: A review. *Big Earth Data* **2018**, *2*, 108–129. [\[CrossRef\]](#)
5. He, W.; Yokoya, N. Multi-Temporal Sentinel-1 and-2 Data Fusion for Optical Image Simulation. *ISPRS Int. J. Geo-Inf.* **2018**, *7*, 389. [\[CrossRef\]](#)
6. Tan, Z.; Yue, P.; Di, L.; Tang, J. Deriving high spatiotemporal remote sensing images using deep convolutional network. *Remote Sens.* **2018**, *10*, 1066. [\[CrossRef\]](#)
7. Ghamisi, P.; Rasti, B.; Yokoya, N.; Wang, Q.; Hofle, B.; Bruzzone, L.; Bovolo, F.; Chi, M.; Anders, K.; Gloaguen, R.; et al. Multisource and multitemporal data fusion in remote sensing: A comprehensive review of the state of the art. *IEEE Geosci. Remote Sens. Mag.* **2019**, *7*, 6–39. [\[CrossRef\]](#)
8. Zhuo, W.; Huang, J.; Li, L.; Zhang, X.; Ma, H.; Gao, X.; Huang, H.; Xu, B.; Xiao, X. Assimilating soil moisture retrieved from Sentinel-1 and Sentinel-2 data into WOFOST model to improve winter wheat yield estimation. *Remote Sens.* **2019**, *11*, 1618. [\[CrossRef\]](#)
9. Huang, J.; Sedano, F.; Huang, Y.; Ma, H.; Li, X.; Liang, S.; Tian, L.; Zhang, X.; Fan, J.; Wu, W. Assimilating a synthetic Kalman filter leaf area index series into the WOFOST model to improve regional winter wheat yield estimation. *Agric. For. Meteorol.* **2016**, *216*, 188–202. [\[CrossRef\]](#)
10. Huang, J.; Tian, L.; Liang, S.; Ma, H.; Becker-Reshef, I.; Huang, Y.; Su, W.; Zhang, X.; Zhu, D.; Wu, W. Improving winter wheat yield estimation by assimilation of the leaf area index from Landsat TM and MODIS data into the WOFOST model. *Agric. For. Meteorol.* **2015**, *204*, 106–121. [\[CrossRef\]](#)
11. Lewis, A.; Oliver, S.; Lymburner, L.; Evans, B.; Wyborn, L.; Mueller, N.; Raevksi, G.; Hooke, J.; Woodcock, R.; Sixsmith, J.; et al. The Australian geoscience data cube—foundations and lessons learned. *Remote Sens. Environ.* **2017**, *202*, 276–292. [\[CrossRef\]](#)
12. Yao, X.; Li, G.; Xia, J.; Ben, J.; Cao, Q.; Zhao, L.; Ma, Y.; Zhang, L.; Zhu, D. Enabling the Big Earth Observation Data via Cloud Computing and DGGS: Opportunities and Challenges. *Remote Sens.* **2020**, *12*, 62. [\[CrossRef\]](#)
13. Ye, S.; Liu, D.; Yao, X.; Tang, H.; Xiong, Q.; Zhuo, W.; Du, Z.; Huang, J.; Su, W.; Shen, S.; et al. RDCRMG: A Raster Dataset Clean & Reconstitution Multi-Grid Architecture for Remote Sensing Monitoring of Vegetation Dryness. *Remote Sens.* **2018**, *10*, 1376.
14. Han, D.; Stroulia, E. Hgrid: A data model for large geospatial data sets in hbase. In Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing, Santa Clara, CA, USA, 28 June–3 July 2013; pp. 910–917.
15. Ye, S. Research on Application of Remote Sensing Tupu-Take Monitoring of Meteorological Disaster for Example. Ph.D. Thesis, China Agricultural University, Beijing, China, 2016.
16. Zhou, M.; Chen, J.; Gong, J. A pole-oriented discrete global grid system: Quaternary quadrangle mesh. *Comput. Geosci.* **2013**, *61*, 133–143. [\[CrossRef\]](#)
17. Dutton, G. Universal geospatial data exchange via global hierarchical coordinates. In Proceedings of the International Conference on Discrete Global Grids, Santa Barbara, CA, USA, 26–28 March 2000; Volume 3, pp. 1–15.
18. Goodchild, M.F.; Guo, H.; Annoni, A.; Bian, L.; De Bie, K.; Campbell, F.; Craglia, M.; Ehlers, M.; Van Genderen, J.; Jackson, D.; et al. Next-generation digital earth. *Proc. Natl. Acad. Sci. USA* **2012**, *109*, 11088–11094. [\[CrossRef\]](#)
19. Cheng, C.; Song, X.; Zhou, C. Generic cumulative annular bucket histogram for spatial selectivity estimation of spatial database management system. *Int. J. Geogr. Inf. Sci.* **2013**, *27*, 339–362. [\[CrossRef\]](#)
20. Lukatela, H. Hipparchus. Data Structure: Points, Lines and Regions in Spherical Voronoi Grid. *Proceedings Auto-Carto.* **1989**, *9*, 164–170.
21. Wang, L.; Zhao, X.; Zhao, L.; Yin, N. Multi-level QTM Based Algorithm for Generating Spherical Voronoi Diagram. *Geomat. Inf. Sci. Wuhan Univ.* **2015**, *40*, 1111–1115.

22. Li, D.; Shao, Z. Spatial information multi-grid and its functions. *Geospat. Inf.* **2005**, *3*, 1–5.
23. Li, D.R.; Xiao, Z.F.; Zhu, X.Y.; Gong, J.Y. Research on grid division and encoding of spatial information multi-grids. *Acta Geod. Cartogr. Sin.* **2006**, *1*, 52–56.
24. Li, D.; Shao, Z.; Zhu, X.; Zhu, Y. From digital map to spatial information multi-grid. In Proceedings of the 2004 IEEE International Geoscience and Remote Sensing Symposium (IGARSS 2004), Anchorage, AK, USA, 20–24 September 2004; Volume 5, pp. 2933–2936.
25. Bjørke, J.T.; Grytten, J.K.; Hæger, M.; Nilsen, S. A Global Grid Model Based on "Constant Area" Quadrilaterals. *ScanGIS Citeaser* **2003**, *3*, 238–250.
26. Bjørke, J.T.; Nilsen, S. Examination of a constant-area quadrilateral grid in representation of global digital elevation models. *Int. J. Geogr. Inf. Sci.* **2004**, *18*, 653–664. [[CrossRef](#)]
27. Ghemawat, S.; Gobiuff, H.; Leung, S.T. The Google file system. In Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, Bolton Landing, NY, USA, 19–22 October 2003.
28. Palankar, M.R.; Iamnitchi, A.; Ripeanu, M.; Garfinkel, S. Amazon S3 for science grids: A viable solution? In Proceedings of the 2008 International Workshop on Data-Aware Distributed Computing, Boston, MA, USA, 25 June 2008; pp. 55–64.
29. Eldawy, A.; Mokbel, M.F. Spatialhadoop: A mapreduce framework for spatial data. In Proceedings of the 2015 IEEE 31st International Conference on Data Engineering, Seoul, Korea, 13–17 April 2015; pp. 1352–1363.
30. Alarabi, L.; Mokbel, M.F.; Musleh, M. St-hadoop: A mapreduce framework for spatio-temporal data. *GeoInformatica* **2018**, *22*, 785–813. [[CrossRef](#)]
31. Borthakur, D. The hadoop distributed file system: Architecture and design. *Hadoop Proj. Website* **2007**, *11*, 21.
32. Liu, X.; Han, J.; Zhong, Y.; Han, C.; He, X. Implementing WebGIS on Hadoop: A case study of improving small file I/O performance on HDFS. In Proceedings of the 2009 IEEE International Conference on Cluster Computing and Workshops, New Orleans, LA, USA, 31 August–4 September 2009; pp. 1–8.
33. Khetrpal, A.; Ganesh, V. *HBase and Hypertable for Large Scale Distributed Storage Systems*; Department of Computer Science, Purdue University: West Lafayette, IN, USA, 2006; Volume 10.
34. Apache HBase. The Apache Software Foundation. 2012. Available online: <http://hadoop.apache.org> (accessed on 8 August 2020).
35. Kaplanis, A.; Kendea, M.; Sioutas, S.; Makris, C.; Tzimas, G. HB+ tree: Use hadoop and HBase even your data isn't that big. In Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, 13–17 April 2015; pp. 973–980.
36. Team, A.H. *Apache Hbase Reference Guide*; Apache, Version; 2016; Volume 2. Available online: <https://hbase.apache.org/book.html> (accessed on 8 August 2020).
37. Liu, Y.; Chen, B.; He, W.; Fang, Y. Massive image data management using HBase and MapReduce. In Proceedings of the 2013 21st International Conference on Geoinformatics, Kaifeng, China, 20–22 June 2013; pp. 1–5.
38. Wang, L.; Cheng, C.; Wu, S.; Wu, F.; Teng, W. Massive remote sensing image data management based on HBase and GeoSOT. In Proceedings of the 2015 IEEE International Geoscience and Remote Sensing Symposium (IGARSS), Milan, Italy, 26–31 July 2015; pp. 4558–4561.
39. Nishimura, S.; Das, S.; Agrawal, D.; El Abbadi, A. Md-hbase: A scalable multi-dimensional data infrastructure for location aware services. In Proceedings of the 2011 IEEE 12th International Conference on Mobile Data Management, Lulea, Sweden, 6–9 June 2011; Volume 1; pp. 7–16.
40. Wang, L.; Chen, B.; Liu, Y. Distributed storage and index of vector spatial data based on HBase. In Proceedings of the 2013 21st international conference on geoinformatics, Kaifeng, China, 20–22 June 2013; pp. 1–5.

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).