Polish Cartographical Review
Vol. 54, 2022, pp. 11–22
DOI: 10.2478/pcr-2022-0002
Received: 21.03.2022
Accepted: 25.04.2022

MICHAŁ WYSZOMIRSKI
Warsaw University of Technology
Faculty of Geodesy and Cartography
Warsaw, Poland
orcid.org/0000-0002-5407-0536; michal.wyszomirski@pw.edu.pl

Analysis of the possibility of using key-value store NoSQL databases for IFC data processing in the BIM-GIS integration process

Abstract. The article discusses the possibility of using Redis key-value NoSQL database to process building data in different BIM-GIS integration solutions. Whichever data integration model is adopted, it will require an efficient serving of building data in Industry Foundation Classes (IFC) format. The author proposed a method of processing building data in the Redis database to support the process of feeding IFC data to his own concept of an integrated BIM-GIS database. However, other approaches to BIM-GIS integration, including the import of IFC data to CityGML, or the construction of an integrated BIM-GIS solution based on data integration at the application server level or client application in client-server environments, also require an efficient IFC data serving mechanism. This article describes three methods of storing IFC data in a Redis database using different data types and formats. The author conducted performance tests of the proposed methods in the processing of fourteen test BIM models. The article contains detailed results of the model processing tests in the Redis database.

Keywords: IFC, Redis, NoSQL, key-value store, BIM-GIS integration

1. Introduction

Menno-Jan Kraak in (Kraak & Ormeling, 2021) discusses the demand for sophisticated geospatial data presentation for continuously increasing amount of spatial data: "The ever more detailed satellite imagery available, the increasing number of sensor networks and new techniques for analyzing textual sources with spatial references like geoparsing all lead to highly varied big data, characterized by large volumes of data, coming available with high velocity." It clearly shows that amount of spatially enabled data is growing, and on the other hand, demand for data from various sources in spatial analyses and presentations is growing too. It leads to spatial big data solutions where data from multiple sources is stored and analyzed (Leszczynski & Crampton, 2016; Shekhar et al., 2012; Yaragal, 2018). In addition, city planners and people trying to build a city or an infrastructure project that connects a city want to visualize the whole thing in context. They want to see the as-planned, the as-designed, and as-built state in a dynamic, intelligent, useful, and persistent city model (Esri, 2019). When considering roads, bridges, waste treatment plants, gas lines, etc., it is not enough to consider their location and relationship. A broader context needs to be considered, including environmental factors such as air quality, solar and wind power capability, storm and flood hazards, fire spread, aviation noise nuisance, soil quality, and much more (Esri, 2022a; Esri, 2022b). Taking this into account, Autodesk, the leader in CAD[1] and BIM[2] software, and Esri, the leader

[1] Computer-Aided Design – computer software for creating, modifying, and analyzing engineering projects.

[2] Building Information Modeling – a type of computer software and management process of the building lifecycle. International standard regulated by ISO 19650.

in GIS[3] software, established a cooperation to put GIS and BIM data at the center of projects. The goal is an integrated and collaborative workflow that improves understanding of projects in context, reduces inefficiencies, and delivers a more sustainable and resilient infrastructure (Autodesk Inc., 2020). Both companies propose a holistic view, where GIS professionals and AEC[4] professionals integrate their unique perspectives of infrastructure: the GIS professional provides insight into the natural and built environment in a larger context. The AEC professionals provide insight into detailed infrastructure assets (Esri, 2022a; Esri, 2022b). It enables the seamless GIS and BIM data flow with a shared environment where GIS professionals and designers/engineers can collaborate across the project life cycle by integrating GIS and BIM platforms (Esri, 2022a; Esri, 2022b). The ESRI-Autodesk integration focuses on extending the functionality of commercial software to exchange data. ArcGIS can import BIM data prepared by Autodesk software, AutoCAD Civil, and Revit can use geospatial data from ArcGIS (Kuehne & Hodge, 2022).

At the same time, ISO works on integrating BIM and GIS data standards. ISO/TR 23262:2021 investigates barriers and proposes solutions to improve interoperability between geospatial and BIM domains (International Organization for Standardization, 2021a). ISO/TS 19166:2021 defines the conceptual framework and mechanisms for mapping information elements from BIM to GIS to access the required information based on specific user requirements (International Organization for Standardization, 2021b). This article focuses on the BIM-GIS integration process based on open standards and open--source software.

Building Information Modeling (BIM) is the process of managing and coordinating work related to the life cycle of a building using a three-dimensional model. The AEC industry uses BIM technology during design and execution works, and after its completion, it submits the BIM model to the manager, who uses it to manage the property. At the same time, the model is updated as part of management work. Since BIM is a constantly updated building model containing, in addition to graphic data, also descriptive data, and technical documentation, it is a valuable source of data about the facility. It can be used to build Smart City solutions that integrate geospatial data from GIS systems with interior data, including technical and operational data of buildings from BIM systems. Integrating BIM and GIS data faces several challenges, including technological differences and utterly different data models. However, resolving these barriers brings tangible benefits for both parties. GIS systems obtain a source of valuable data on individual buildings. BIM systems gain access to GIS data and technologies. Commercial agreement between Esri and Autodesk and ISO standards are relatively new attempts to integration of BIM and GIS data. Various scenarios of data integration from both systems are considered in scientific projects for a long time (Biljecki et al., 2015; Fosu et al., 2015; Guyo et al., 2021; Karimi & Iordanova, 2021; Pauwels et al., 2017). In each of them, a necessary element is the method of processing BIM data stored in the IFC[5] model in a text file in this format. The IFC is a standard data model and file format for storing and processign building data. It is widely used as exchange format of the BIM related data. The author developed and described his own method of BIM and GIS data integration (Wyszomirski & Gotlib, 2020), in which he used a NoSQL[6] database to preprocess IFC data. This allowed to significantly improve and accelerate the process of BIM data processing in the IFC format.

During the experiments with the integration of BIM and GIS data, some problems were observed with the efficiency of data processing procedures because of the long time it took to read a single IFC object from an IFC file. In the test environment (workstation equipped with a six-core processor, 32 GB RAM, Windows operating system) in the case of a model consisting of about 3 million objects, the reading

[3] Geographical Information System – (in the scope of this article) a type of computer software to store, process, analyze, and visualize geospatial data. International standard regulated by multiple ISO standards estabilished by ISO/TC 211 (Geographic information/Geomatics) Committee.

[4] Architectural, Engineering, and Construction industry.

[5] Industry Foundation Classes – data model and file format intended to describe architectural and construction industry data. International standard regulated by ISO 16739 (International Organization for Standardization, 2013).

[6] NoSQL – generic term for all distributed databases whose data model is optimized for unstructured data processing.

time for a single IFC object was about 1.5 seconds, with a sequential search for objects in a text file. The method of reading data in the IFC model requires that even several hundred IFC objects of about 20 different IFC classes are used to describe a single building element, such as a wall, a ceiling, or a room. The process of reading the complete dataset for a single building element can, therefore, take up to several minutes. This time can be significantly shortened by preloading the entire IFC model into RAM before processing it. However, this solution was rejected by the author because the concept of BIM-GIS integrated solution assumes work in an environment integrating BIM and GIS data at the scale of at least the entire city, with thousands of BIM models being processed simultaneously. Loading many such models into RAM while supplying BIM data into the database was considered unacceptable. Therefore, an additional attempt was made to use the NoSQL database as an extra stage of data processing. The use of NoSQL databases provided very quick access to individual elements of the building model. Therefore, it was proposed to introduce an additional stage to the process of IFC data import into an integrated BIM-GIS database, which consisted of rewriting the contents of the IFC file into the NoSQL database (of key-value store type) and executing the necessary read operations within it. Redis database was used for the experiment. Thanks to this operation, very fast access to all elements of the IFC model was obtained. With this approach, the average time taken to read a single object from an IFC file was one millisecond. The NoSQL database can be used to obtain, in a very short time, basic information about the modeled object, such as the number of objects of particular IFC classes in the processed model, the number of buildings, floors, and rooms in the model, and other structural elements, as well as the location of the model. It allows performing statistical analyses of the objects stored in it, which can be used to check the correctness of importing IFC data into an integrated BIM-GIS database and provide additional information about the model.

As part of the research work, the author used three different data processing methods using the NoSQL database working in the key-value model. The most popular database of this type was selected for the test – Redis. The test results, including the performance comparison, were described, and the most efficient solution was proposed based on the results.

2. Materials and methods

The IFC format and model[7] are dedicated to the exchange of information with other IT systems. It is promoted as a universal data model supporting data exchange and building lifecycle support (Solibri, 2017). It is a text-based, object-modeled file format developed to facilitate interoperability in the architectural and structural engineering (AEC) industries. IFC is defined in ISO 16739: 2013. The definition of the IFC data model is described in the EXPRESS language[8], an industrial object data modeling language. The IFC model provides a set of definitions for all types of object elements found in construction, and the IFC is the structure for storing these definitions in text data files. The IFC definition allows for several file formats for data transfer:

1. STEP[9] IFC – the IFC model can be saved in a STEP file, the format of which is sanctioned by the ISO 10303-21 standard (International Organization for Standardization, 2016).

2. IFCXML – the IFC model can be saved in an XML[10] file, the format of which is a subset of Standard Generalized Markup Language (SGML) regulated by ISO 8879 (International Organization for Standardization, 1986) and sanctioned by the W3C specification (W3C, 2015).

[7] IFC (Industry Foundation Classes) was created by the Industry Alliance for Interoperability, established in 1994 by Autodesk. In 1997, the name of the consortium was changed to International Alliance for Interoperability in order to emphasize the emphasis on cooperation between many industries, both in the development of the standard itself and in its use. Currently, the consortium is called buildingSMART.

[8] The EXPRESS language is described in the ISO 10303 standard "Standard for the Exchange of Product model STEP" and standardized in 1994 by ISO 10303-11 (International Organization for Standardization, 2004).

[9] STandard for the Exchange of Product model data – data model and file format for representing and exchanging product manufacturing information. International standard regulated by ISO 10303.

[10] Extensible Markup Language – a markup language and file format for storing and transmitting data. International standard regulated by ISO 20022.

3. IFCZIP – is an IFC file in XML format packed with the Deflate compression algorithm based on the Lempel-Ziv 77 (LZ77) dictionary data compression streaming method and Huffman coding.

Transfer of BIM data with IFC files presents many problems: the files are large, and processing is time-consuming.

Redis is currently the most popular key-value database[11] by db-engines.com (DB-Engines, 2022). In terms of compliance with the CAP theorem[12], this database operates in the CP model (consistency and partition tolerance without ensuring continuous data availability). Salvatore Sanfilippo wrote the system in 2006 in the C language (Da Silva & Tavares, 2015). Redis stands for ‘REmote DIctionary Server’. Currently, work on the development of the Redis database is financed by Redis Labs, and the database is available under a BSD license[13] (Redis Labs, 2016). Redis is a data structure store that is stored in the operating memory of a server (Seguin, 2015). It is commonly used as a database, cache, and message broker (Haber, 2017). Redis has its own query language. In addition, libraries allowing access to the Redis database are available for the most popular programming languages. Redis is a data structure server with an in-memory dataset optimized for speed of data access. It is called a data structure server and not simply a key-value store because Redis implements data structures allowing assigning different data structures to keys stored in the database. This combination of flexibility and speed makes Redis the ideal tool for many applications (Macedo & Oliveira, 2011).

The main advantage of NoSQL is that there is no concept of normalization. This is the reason why NoSQL database gives more performance when seen against a normalized SQL database. There is a trade-off in that data consistency is sacrificed in the NoSQL database, but the benefits achieved in doing so are higher. Clearly, NoSQL databases are built with one central feature, which is performance. So, to achieve performance, data consistency and reliability are sacrificed at various levels (Chinnachamy, 2014).

In NoSQL databases using the key-value model, both key and value are typically a string. In the Redis database, the value is not limited to a simple string, but can also contain more complex data structures (Redis Labs, 2017). The list of available data types in the Redis database includes (Da Silva & Tavares, 2015):

• Strings, that can also be treated as numbers (integer, float), texts (unformatted or in XML, JSON[14], HTML[15] format), or as binary strings (video files, graphic files, and audio files) depending on how applications use them. The value of a String field cannot exceed 512 MB.

• Lists that can be thought of as simple collections, stacks, or queues.

• Hash tables are indexed arrays that allow searching for a value using an index based on the given key, the hash function determines the index of the value in the table.

• Sets, which are an unordered collection of unique values. Internally, the set is implemented with hash arrays.

• Sorted sets are very similar to sets, but each value is assigned a weight by which the set is sorted.

• HyperLogLog, which is a special data type that allows the execution of the HyperLogLog algorithm to determine the number of unique values in a set. This algorithm is based on a probabilistic determination of the number of elements. It allows to quickly determine the number of elements of a set with a high probability, using a very small amount of operational memory (Flajolet et al., 2007).

In IFC files, data in the form of a list of objects is identified by an identifier assigned to each object that is unique in the scale of the entire IFC file. Later in this article, the internal identifier is referred as ifcid. The ifcid identifier in the IFC file is a numeric value preceded by a # character. An example, one line of the IFC file

[11] Key-value database – a database that uses a key-value data model to store data.

[12] Consistency, Availability, Partition tolerance – theorem is also known as Brewer’s Theorem; it states that any distributed data store can only provide two of the three attributes: Consistency, Availability, Partition tolerance.

[13] BSD License (Berkeley Software Distribution License) – a license to distribute software under the Free Software Principles of the University of California, Berkeley.

[14] JavaScript Object Notation – an open standard file format and data interchange format. International standard regulated by ISO 21778.

[15] HyperText Markup Language – a standard markup language for documents designed to be displayed in a Internet browser. International standard regulated by ISO 8859.

Tab. 1. IfcBuilding class attributes (from the IFC model) including attributes inherited from parent classes

Attribute	Type
GlobalId	IfcGloballyUniqueId (STRING)
OwnerHistory	IfcOwnerHistory (ENTITY)
Name	IfcLabel (STRING)
Description	IfcText (STRING)
ObjectType	IfcLabel (STRING)
ObjectPlacement	IfcObjectPlacement (ENTITY)
Representation	IfcProductRepresentation (ENTITY)
LongName	IfcLabel (STRING)
CompositionType	IfcElementCompositionEnum (ENUM)
ElevationOfRefHeight	IfcLengthMeasure (REAL)
ElevationOfTerrain	IfcLengthMeasure (REAL)
BuildingAddress	IfcPostalAddress (ENTITY)

describing an object of IfcBuilding class looks like this:

```
#62=IFCBUILDING(‘3cNcsqqFH6jAJu-
Rvg0K0Rk’,#13,’Default Building’
,$,$,#60,$,$, .ELEMENT.,$,$,$);
```

In this example, #62 is the local ifcid of the IFC object, while the text after the equal sign (=) is the description of the IFC object consisting of its class name (IfcBuilding – building in the IFC model) and its attributes that are defined in the IFC schema. The list of attribute names of the IfcBuilding class object is presented in table 1.

The object of IfcBuilding class is related to the objects of other classes that define the features of the building. For example, the Representation attribute points to an IfcProduct-Representation object that aggregates all the geometric elements that make up the geometric representation of the object. Object descriptive attributes are assigned to an IfcBuilding object through an IfcRelAggregates aggregate object that identifies the IfcPropertySet objects assigned to an IfcBuilding object that contains sets of attributes. The stories, which are a component element in the spatial structure of the IfcBuilding object, are assigned to the IfcBuilding class object through the IfcRelAggregates class object being an aggregate element which identifies the IfcBuildingStorey class objects assigned to the IfcBuilding object (Gotlib & Wyszomirski, 2017).

Redis allows writing values assigned to a given key and reading these values by key. Several

Tab. 2. An example of storing IFC data for an object of IfcBuilding class in the Redis database using a string value

Key	Value
#62	IFCBUILDING(‚3cNcsqqFH6jAJuRvg0K0Rk’,#13,’Default Building’,$,$,#60, $,$,.ELEMENT.,$,$,$);

Tab. 3. An example of storing IFC data for an object of IfcBuilding class in the Redis database using the hash table

Key	Value	
#62	IfcClass	IFCBUILDING
	GlobalId	‚3cNcsqqFH6jAJuRvg0K0Rk’
	OwnerHistory	#13
	Name	‚Default Building’
	Description	$
	ObjectType	$
	ObjectPlacement	#60
	Representation	$
	LongName	$
	CompositionType	.ELEMENT.
	ElevationOfRefHeight	$
	ElevationOfTerrain	$
	BuildingAddress	$

data types available in the Redis database can be used to store IFC data, three of which seem to be exceptionally well suited to the IFC model:

• Simple key of the string type, the value stored in the string type.

• Simple key of the string type, the value stored in the hash table type.

• Complex key of the string type, the value stored in the string type.

Using the write method, where there is a simple key of the string type and a value stored in the string type, an example IfcBuilding object with ifcid equal to #62 will look like presented in table 2.

Using the write method, where there is a simple key of the string type and a value stored in the hash table type, an example IfcBuilding object with ifcid #62 will look as shown in table 3.

Using the write method, where there is a simple key of the string type and a value stored in the hash table type, an example IfcBuilding object with ifcid #62 will look as shown in table 4.

2.1. Test environment

The experiments were conducted using an environment consisting of a workstation equipped with a six-core processor, 32 GB RAM, a Windows operating system, and a Redis database server equipped with a quad-core processor, 8 GB RAM, and CentOS 7 operating system. The database server would run as a virtual machine running on a host running Microsoft Hyper-V Server. Originally, the Redis database server was installed on a virtual machine with the Windows Server operating system. However, since the test process took about a week of continuous operation, an unexpected restart of the Windows Server operating system caused by Windows Update interrupted its operation, making Windows Server unreliable to such tests. The next step was to transfer the test environment to the CentOS system.

2.2. Test data

Fourteen IFC models of different sizes and levels of detail and representing different buildings or elements of the buildings were used for the tests. Some models

Tab. 4. An example of storing IFC data for an object of IfcBuilding class in the Redis database using composite keys and string values

Key	Value
#62:IfcClass	IFCBUILDING
#62:GlobalId	,3cNcsqqFH6jAJuRvg0K0Rk’
#62:OwnerHistory	#13
#62:Name	,Default Building’
#62:Description	$
#62:ObjectType	$
#62:ObjectPlacement	#60
#62:Representation	$
#62:LongName	$
#62:CompositionType	.ELEMENT.
#62:ElevationOfRefHeight	$
#62:ElevationOfTerrain	$
#62:BuildingAddress	$

Tab. 5. Summary of basic data on all models used in the test, including the number of buildings, floors, rooms, doors, and windows in each model and the total number of all objects in the model

Model IFC	Number of					Object count
	buildings	floors	rooms	doors	windows	
01.ifc	1	14	1383	718	1132	3140740
02.ifc	1	8	480	559	385	2651693
03.ifc	1	8	258	265	349	2094772
04.ifc	1	8	258	265	349	1897218
05.ifc	1	13	1661	1896	702	1421804
06.ifc	1	9	2612	2623	3560	1204965
07.ifc	1	7	421	405	1589	726390
08.ifc	1	6	656	1068	160	526341
09.ifc	1	7	199	208	122	467901
10.ifc	1	10	136	152	170	329855
11.ifc	1	10	136	149	169	179709
12.ifc	1	3	71	59	25	160141
13.ifc	1	1	39	35	20	40094
14.ifc	1	1	0	1	5	2711

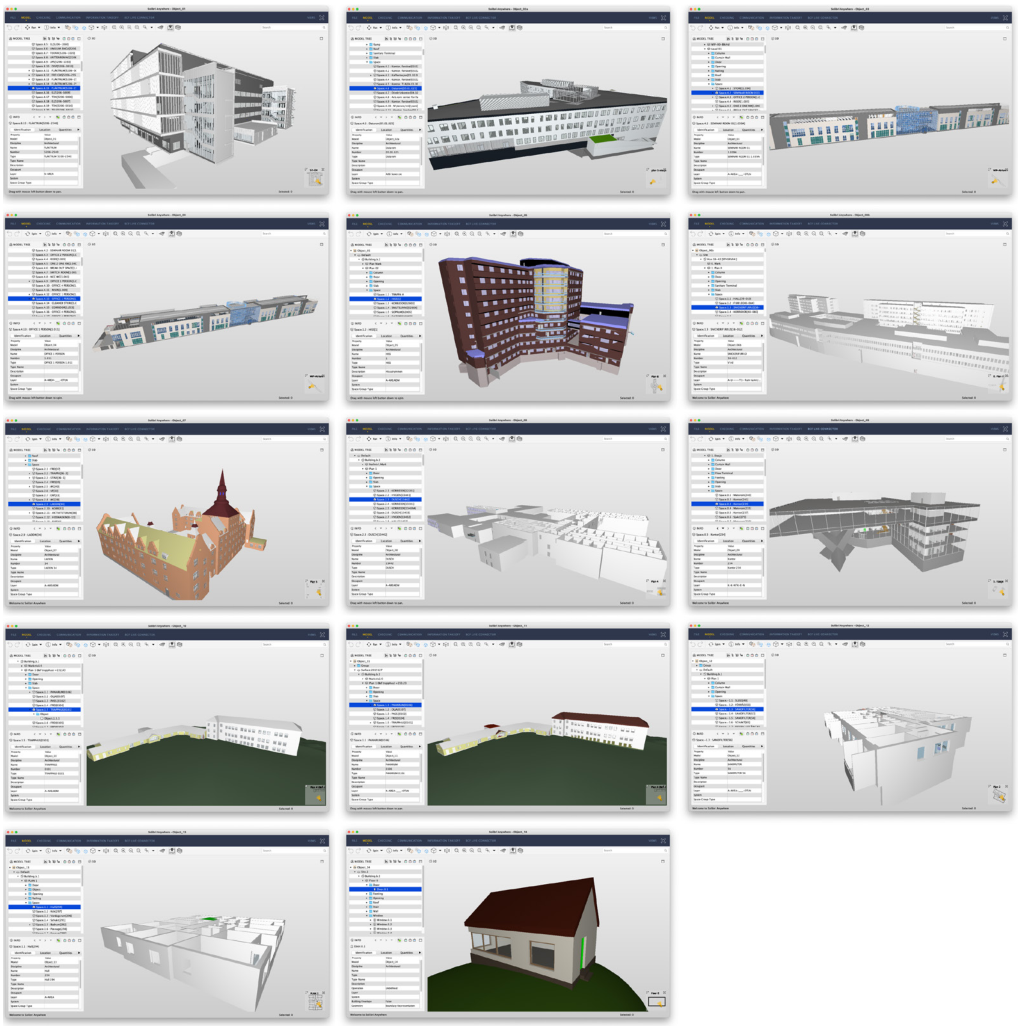
Fig. 1. Visualization of all models used in the test prepared in Solibri Anywhere. Top row left to right: Object_01, Object_02, Object_03; second row left to right: Object_04, Object_05, Object_06; third row left to right: Object_07, Object_08, Object_09; fourth row left to right: Object_10, Object_11, Object_12; last row left to right: Object_13, Object_14

represent existing buildings, others are at project stage only. Figure 1 visualizes the models in Solibri Anywhere software.

The smallest IFC model used in the tests was a model of a very simple single-family house. The entire model consisted of 2,711 objects. The largest model in the test was a model of a large office building with 14 floors and 1,383 rooms. The model of this building has 718 doors and 1132 windows. In two cases (Object_3/Object_4 and Object_10/Object_11), two models of the same building were used. However, since they had different sizes and numbers of objects, they were qualified for the test as different examples of IFC models. A summary of basic data on all models used in the test, including the number of buildings, floors, rooms, doors, and windows in each model, and the total number of all objects in the model, is presented in table 5.

Before starting the actual tests of data processing with the use of the Redis database, a test was performed consisting in comparing the number of objects of individual IFC classes in each of the models. This operation revealed the details of each model, including their complexity, and allowed further testing to be designed. Table 6 summarizes the number of objects and preprocessing times for each of the IFC models under test.

2.3. Test process

The tests were carried out using scripts written in the Python programming language. Four test cases were planned and then performed:

• Search for an object with a given ifcid in an IFC text file.

• Search for the object stored in the Redis database in the case of writing data in the form of a character string assigned to a simple key containing only the IFC identifier (ifcid).

• Search for the same object stored in the Redis database when saving data in the form of a hash table.

• Search for the same object saved in the Redis database in the case of saving data in the form of separate strings of characters corresponding to individual object attributes, assigned to complex keys consisting of ifcid and the name of the attribute. During the test, it was decided to measure the access time to all object attributes stored in separate keys in the Redis database. It is worth noting, however, that when a single IFC object is stored as multiple key--value pairs in the database, there is easy access to selected object attributes.

3. Results

The performance test of the IFC data processing process using the NoSQL key-value database has been divided into several stages. The first stage included the process of rewriting the content of IFC files to the Redis database, considering the three proposed data processing models: text data, hash tables, and the use of a complex key. The test results of rewriting the data to the database are summarized in table 7. The duration of the process is presented in seconds. As it can be seen, the time to rewrite the smallest model (Object_14.ifc) is from less than a second for a text type to more than two seconds for models using the hash table type and complex keys. The rewrite time for the largest

Tab. 6. Summary of the number of objects and preprocessing times for each IFC model tested

Model IFC	Object count	Preprocessing time [s]
01.ifc	3140740	5,53271
02.ifc	2651693	4,45943
03.ifc	2094772	3,52175
04.ifc	1897218	3,26385
05.ifc	1421804	2,59090
06.ifc	1204965	2,18576
07.ifc	726390	1,24690
08.ifc	526341	0,95293
09.ifc	467901	0,79596
10.ifc	329855	0,55597
11.ifc	179709	0,29606
12.ifc	160141	0,25905
13.ifc	40094	0,06506
14.ifc	2711	0,00400

Tab. 7. List of IFC models rewriting times from text files to the database, considering three IFC data models in the Redis database: as a string, hash table, and compound keys

Model IFC	Rewrite file [s]		
	string	hash table	complex key
01.ifc	993,88681	3946,96158	3913,15126
02.ifc	851,85908	2215,50767	2233,54447
03.ifc	658,24368	2041,04050	2058,39088
04.ifc	602,91097	1774,04592	1759,16744
05.ifc	451,36611	1715,56905	2039,94314
06.ifc	384,88137	1328,36712	1319,42996
07.ifc	230,97379	837,72939	843,46118
08.ifc	167,25845	712,85931	708,06758
09.ifc	150,78017	460,40673	455,09957
10.ifc	106,16182	300,98696	301,87119
11.ifc	58,20541	204,50587	204,45121
12.ifc	51,30364	136,93056	138,47969
13.ifc	12,92979	44,55429	44,10076
14.ifc	0,86345	2,45046	2,34236

model (Object_01.ifc) ranges from 16 minutes to over an hour, depending on the data model used.

The next stage of the test was to determine the access time to a single object in the model. When reading data sequentially from a text file, the access time to data at the beginning of the file is much shorter than for data at the end of the file. This means that it is needed to read almost the entire file to get to the items at the end of the file. In the case of a database, the access time to the data should be the same, regardless of whether they were initially written at the beginning of the file or at its end. To determine the differences in access times to the elements of the IFC model, a test was proposed. It measures the access time to the first object, the last object, and the object that is in the middle of the IFC file content. To this end, the ifcid of the first, middle, and last objects were specified. A summary of the numbers of these facilities is presented in table 8.

Then, the first, middle, and last objects read time tests were performed in each model, considering different data storage models in the database. For comparison, a test of reading the same objects from IFC files was also performed. The list of reading times for the first object in each of the test models is presented in table 9. As can be seen, the times of reading data from the database are very similar for the three data models and oscillate around one millisecond. In the case of a text file, the reading time of the first object ranges from 1 millisecond to 2.5 milliseconds. Thus, it can be assumed that the access time to the first object is comparable for the file and the database.

The summary of the reading times of the middle object in each of the test models is presented in table 10. As can be seen, the times of reading data from the database are very similar to each other for the three data models and to the test results with the first objects in the models and oscillate within 1 millisecond. In the case of a text file, the reading time of the first object ranges from 2 milliseconds for the smallest model (Object_14) to 1,7 seconds for the largest model (Object_1).

The summary of the reading times of the last object in each of the test models is presented in table 11. As it can be seen, the times of reading data from the database for the three data models differ slightly. For most models, the reading

Tab. 8. List of ifcid of first, middle, and last objects in test models

Model IFC	Value of IfcId			Object count
	first	middle	last	
01.ifc	1	2887599	5775198	3140740
02.ifc	1	5019298	10038596	2651693
03.ifc	1	1883932	3767864	2094772
04.ifc	1	1689045	3378090	1897218
05.ifc	1	715823	1431646	1421804
06.ifc	1	2379114	4758228	1204965
07.ifc	1	619285	1238570	726390
08.ifc	1	264894	529787	526341
09.ifc	1	418938	837875	467901
10.ifc	1	289098	578196	329855
11.ifc	1	158201	316402	179709
12.ifc	1	140622	281244	160141
13.ifc	1	35636	71272	40094
14.ifc	1	1356	2711	2711

Tab. 9. List of reading times for the first object in test models from the database in each of the three proposed data models and from a text file

Model IFC	Search time – first object [s]			
	string	hash table	complex key	file
01.ifc	0,00073	0,02092	0,00073	0,00146
02.ifc	0,00100	0,00325	0,00037	0,00219
03.ifc	0,00090	0,00103	0,00064	0,00172
04.ifc	0,00089	0,00102	0,00037	0,00202
05.ifc	0,00063	0,00100	0,00055	0,00248
06.ifc	0,00094	0,00100	0,00055	0,00220
07.ifc	0,00080	0,00091	0,00045	0,00202
08.ifc	0,00080	0,00101	0,00073	0,00192
09.ifc	0,00054	0,00101	0,00079	0,00136
10.ifc	0,00089	0,00091	0,00055	0,00146
11.ifc	0,00063	0,00102	0,00037	0,00182
12.ifc	0,00266	0,00177	0,00074	0,00145
13.ifc	0,00072	0,00100	0,00064	0,00201
14.ifc	0,00072	0,00100	0,00054	0,00155

Tab. 10. List of middle objects reading times in test models from the database in each of the three proposed data models and from a text file

Model IFC	Search time – mid object [s]			
	string	hash table	complex key	file
01.ifc	0,00053	0,00072	0,00073	1,73017
02.ifc	0,00081	0,00072	0,00091	1,19712
03.ifc	0,00090	0,00082	0,00018	0,98712
04.ifc	0,00082	0,00091	0,00092	0,85776
05.ifc	0,00036	0,00100	0,00082	0,70924
06.ifc	0,00039	0,00072	0,00072	0,56832
07.ifc	0,00045	0,00064	0,00046	0,35663
08.ifc	0,00062	0,00100	0,00073	0,26228
09.ifc	0,00091	0,00073	0,00064	0,23009
10.ifc	0,00072	0,00109	0,00070	0,15792
11.ifc	0,00064	0,00101	0,00091	0,09083
12.ifc	0,00073	0,00082	0,00082	0,07735
13.ifc	0,00072	0,00082	0,00087	0,02165
14.ifc	0,00091	0,00064	0,00101	0,00273

Tab. 11. List of last objects reading times in test models from the database in each of the three proposed data models and from a text file

Model IFC	Search time – last object [s]			
	string	hash table	complex key	file
01.ifc	0,00341	0,00485	0,00462	3,36207
02.ifc	0,00100	0,00137	0,00090	2,44489
03.ifc	0,00320	0,00429	0,00127	2,03286
04.ifc	0,00597	0,00457	0,00072	1,74784
05.ifc	0,00100	0,00118	0,00073	1,41482
06.ifc	0,00092	0,00109	0,00091	1,14668
07.ifc	0,00711	0,00531	0,00081	0,72424
08.ifc	0,00684	0,01129	0,00065	0,54262
09.ifc	0,00346	0,00529	0,00090	0,46499
10.ifc	0,00154	0,00165	0,00173	0,31496
11.ifc	0,00118	0,00156	0,00147	0,17254
12.ifc	0,00109	0,00137	0,00120	0,15057
13.ifc	0,00146	0,00183	0,00101	0,04038
14.ifc	0,00090	0,00118	0,00091	0,00310

time is still one millisecond, but for some of them, this time has been extended to over five milliseconds (hash table type for the Object_07 model). In the case of a text file, the reading time of the first object ranges from 3 milliseconds for the smallest model (Object_14) to 3.3 seconds for the largest model (Object_1).

4. Discussion

The tests show that a simple search for an IFC model object stored in the Redis database can be up to 1000 times faster than the same operation performed on a text file in the IFC format. The use of the method of saving each IFC object in the database in the form of a key pair containing the ifcid identifier and a value containing the entire description of the object taken from the IFC file and written with a text string allows the object to be read much faster than from a text file. An application using this data model will have to parse the content of each line retrieved from the database in the same way as it is done when reading an IFC file. The use of hash tables is a slightly slower solution, but it allows to get a dictionary of attributes instead of a string. Further processing of the data will therefore be faster because no serialization of the attributes is required anymore, and the reading is done directly into the Python dictionary type. The slowest is processing IFC data with the use of complex keys, which are built in the form of an ifcid identifier enriched with the names of attributes. In this case, the acceleration compared to reading from a text file is lower. However, this method allows reading selected individual IFC object attributes.

5. Conclusion

The Redis database has no mechanisms supporting the search for items in the database. Therefore, the Redis database cannot be used to search for a specific class of IFC objects, which is a common operation when processing IFC files. This operation must be performed on the application side, which reads the data from the database and then processes it. In return, Redis can deliver data very quickly according to the given key, which significantly improves the process of reading the BIM model saved in

IFC. Comparing the reading time of a single object from an IFC text file with the reading time of the same object from the Redis database shows significant differences in favor of the database solution. The main benefit of using Redis to store and preprocess building data in IFC model is flexibility and speed. It allows access to elements of building model in very convenient way fitted to data model used by application and also allows very fast access to IFC data. The method of saving the IFC model in the Redis database – in the form of strings, hash tables, or in the form of compound keys – should be adapted to the requirements of the IFC data processing application. If Redis is to be a tool that accelerates the simple reading of the content of an IFC file, the use of strings is sufficient. However, if the Redis database were to provide more structured data, e.g., allow for the retrieval of selected attributes of IFC objects, the use of hash tables or complex keys is a more beneficial solution.

Literature

Autodesk Inc. (2020). *Autodesk & Esri Collaboration*. https://www.autodesk.com/solutions/bim/hub/autodesk-esri

Biljecki, F., Stoter, J., Ledoux, H., Zlatanova, S., & Çöltekin, A. (2015). Applications of 3D City Models: State of the Art Review. *ISPRS International Journal of Geo-Information, 4*(4), 2842–2889. https://doi.org/10.3390/ijgi4042842

Chinnachamy, A. (2014). *Redis Applied Design Patterns*. Packt Publishing Ltd.

Da Silva, M. D., & Tavares, H. L. (2015). *Redis Essentials*. Packt Publishing Ltd.

DB-Engines. (2022, April). *DB-Engines Ranking – popularity ranking of database management systems*. https://db-engines.com/en/ranking

Esri. (2019). *Esri-Autodesk Partnership*. https://www.esri.com/about/newsroom/arcnews/esri-autodesk-partnership/

Esri. (2022a). *Esri & Autodesk Combining the power of location and desig*n. https://geobim.maps.arcgis.com/apps/Cascade/index.html?appid=e613e7f3157b45c09e9455e79912d2a4

Esri. (2022b). *Esri i Autodesk − współpraca, która łączy GIS i BIM*. https://www.esri.com/pl-pl/about/partners/our-partners/strategic-alliances/autodesk

Flajolet, P., Fusy, É., Gandouet, O., & Meunier, F. (2007). HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm. *Discrete Mathematics & Theoretical Computer Science, DMTCS Proceedings, AH*, 137–156. https://doi.org/10.46298/dmtcs.3545

Fosu, R., Suprabhas, K., Rathore, Z., & Cory, C. (2015). Integration of Building Information Modeling (BIM) and Geographic Information Systems (GIS)−A literature review and future needs. *Proceedings of the 32nd International Conference of CIB W78, Eindhoven, The Netherlands, 27–29 October*, 196–204. http://itc.scix.net/paper/w78-2015-paper-020

Gotlib, D., & Wyszomirski, M. (2017, November). *Ocena możliwości konwersji modeli BIM na modele GIS* [Paper presentation]. XXVII Konferencja Polskiego Towarzystwa Informacji Przestrzennej, Warszawa, Polska.

Guyo, E., Hartmann, T., & Ungureanu, L. (2021). Interoperability between BIM and GIS through open data standards: An overview of current literature. *LDAC2021 − 9th Linked Data in Architecture and Construction Workshop. Luxembourg*. http://ceur-ws.org/Vol-3081/10paper.pdf

Haber, I. (2017). *Redis for Geospatial Data*. Redis Labs.

International Organization for Standardization. (1986, October). *ISO 8879:1986 Information processing − Text and office systems − Standard Generalized Markup Language (SGML)*. https://www.iso.org/standard/16387.html

International Organization for Standardization. (2004, November). *ISO 10303-11:2004 Industrial automation systems and integration − Product data representation and exchange − Part 11: Description methods: The EXPRESS language reference manual*. https://www.iso.org/standard/38047.html

International Organization for Standardization. (2013, April). *ISO 16739:2013 Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries*. https://www.iso.org/standard/51622.html

International Organization for Standardization. (2016, March). *ISO 10303-21:2016 Industrial automation systems and integration − Product data representation and exchange − Part 21: Implementation methods: Clear text encoding of the exchange structure*. https://www.iso.org/standard/63141.html

International Organization for Standardization. (2021, May a). *ISO/TR 23262:2021. GIS (geospatial) / BIM interoperability*. https://www.iso.org/standard/75105.html

International Organization for Standardization. (2021, May b). *ISO/TS 19166:2021. Geographic information − BIM to GIS conceptual mapping (B2GM)*. https://www.iso.org/standard/78899.html

Karimi, S., & Iordanova, I. (2021). Integration of BIM and GIS for Construction Automation, a Systematic Literature Review (SLR) Combining Bibliometric and Qualitative Analysis. *Archives of Computational*

Methods in Engineering, 28, 4573–4594. https://doi.org/10.1007/s11831-021-09545-2

Kraak, M.-J., & Ormeling, F. (2021). *Cartography. Visualization of Geospatial Data*. CRC Press Taylor & Francis Group, LLC.

Kuehne, D., & Hodge, K. (2022). *What's Possible with Esri Autodesk Integration*. https://www.esri.com/content/dam/esrisites/en-us/about/events/media/UC-2019/technical-workshops/tw-6222-911.pdf

Leszczynski, A., & Crampton, J. (2016). Introduction: Spatial Big Data and everyday life. *Big Data & Society*. https://doi.org/10.1177/2053951716661366

Macedo, T., & Oliveira, F. (2011). *Redis Cookbook*. O'Reilly Media, Inc.

Pauwels, P., Zhang, S., & Lee, Y.-C. (2017). Semantic web technologies in AEC industry: A literature overview. *Automation in Construction, 73*, 145–165.

Redis Labs. (2016, September 08). *Redis*. http://redis.io/

Redis Labs. (2017, July 18). *An introduction to Redis data types and abstractions*. https://redis.io/topics/data-types-intro

Seguin, K. (2015). *The Little Redis Book*. https://github.com/karlseguin/the-little-redis-book

Shekhar, S., Gunturi, V. M., Evans, M. R., & Yang, K. (2012). Spatial big-data challenges intersecting mobility and cloud computing. *MobiDE,12: Proceedings of the Eleventh ACM International Workshop on Data Engineering for Wireless and Mobile Access*, 1–6. https://doi.org/10.1145/2258056.2258058

Solibri. (2017, September 24). *About BIM and IFC*. http://www.solibri.com/support/bim-ifc/

Wyszomirski, M., & Gotlib, D. (2020). A Unified Database Solution to Process BIM and GIS Data. *Applied Sciences, 10*(23), 8518. https://doi.org/10.3390/app10238518

Yaragal, S. (2018, February 10). *Big data in GIS environment*. https://www.geospatialworld.net/blogs/big-data-in-gis-environment/

The World Wide Web Consortium (W3C). (2015). *W3C Recommendation. Extensible Markup Language (XML) 1.0 (Fifth Edition)*. https://www.w3.org/TR/xml/