BIG DATA MINING AND ANALYTICS
ISSN 2096-0654 07/15 pp371–398
DOI: 10.26599/BDMA.2023.9020039
Volume 7, Number 2, June 2024

Data Temperature Informed Streaming for Optimising Large-Scale Multi-Tiered Storage

Dominic Davies-Tagg, Ashiq Anjum, Ali Zahir*, Lu Liu,
Muhammad Usman Yaseen, and Nick Antonopoulos

Abstract: Data temperature is a response to the ever-growing amount of data. These data have to be stored, but they have been observed that only a small portion of the data are accessed more frequently at any one time. This leads to the concept of hot and cold data. Cold data can be migrated away from high-performance nodes to free up performance for higher priority data. Existing studies classify hot and cold data primarily on the basis of data age and usage frequency. We present this as a limitation in the current implementation of data temperature. This is due to the fact that age automatically assumes that all new data have priority and that usage is purely reactive. We propose new variables and conditions that influence smarter decision-making on what are hot or cold data and allow greater user control over data location and their movement. We identify new metadata variables and user-defined variables to extend the current data temperature value. We further establish rules and conditions for limiting unnecessary movement of the data, which helps to prevent wasted input output (I/O) costs. We also propose a hybrid algorithm that combines existing variables and new variables and conditions into a single data temperature. The proposed system provides higher accuracy, increases performance, and gives greater user control for optimal positioning of data within multi-tiered storage solutions.

Key words: data temperature; hot and cold data; multi-tiered storage; metadata variable; multi-temperature system

1 Introduction

According to recent studies, we have created over forty zetabytes of data[1], collecting more data in the past few years than that has ever existed before. This volume of data come in a variety of formats, such as millions of tweets sent daily or terabytes generated by genomics sequencing[2]. The problem with storing all these data gets addressed with cheaper hardware and commodity storage solutions such as Hadoop, allowing for significant amounts of storage across a distributed cluster of nodes. Hadoop, although powerful, is not known for high-performance analytical tasks or fast input output (I/O)[3].

Advances in technology have also meant a significant decrease in the price of memory (random access memory (RAM)), but also the development of faster storage hardware such as solid state drive (SSD) (Flash-Memory) and PCIe NVMe SSDs. All of these are faster than traditional hard disks with their slow physical platters. Random access memory and flash

• Dominic Davies-Tagg is with the Department of Computing, University of Derby, Derby, DE22 1GB, UK. E-mail: d.tagg@derby.ac.uk.
• Ashiq Anjum, Ali Zahir, and Lu Liu are with the Department of Informatics, University of Leicester, Leicester, LE1 7RH, UK. E-mail: a.anjum@leicester.ac.uk; sazb1@leicester.ac.uk; l.liu@leicester.ac.uk.
• Muhammad Usman Yaseen is with the Department of Computer Science, COMSATS University Islamabad, Islamabad 45550, Pakistan. E-mail: usman_yaseen@comsats.edu.pk.
• Nick Antonopoulos is with the Edinburgh Napier University, Edinburgh, EH11 4BN, UK. E-mail: N.Antonopoulos@napier.ac.uk.
* To whom correspondence should be addressed.
Manuscript received: 2023-04-24; revised: 2023-07-29; accepted: 2023-12-07

memory technologies have no moving parts, and this in part gives them their much higher I/O speed[4].

The above are viable storage mediums, but the cost compared to commodity storage is exorbitant, so only the most critical and high priority data get used by these solutions. All other data would need offloading to cheaper storage, such as hard disk drive (HDDs) or tapes. Considering the benefits of both the storage across distributed commodity nodes and the expensive high-performance memory storage, it makes sense to combine both into a single storage solution. A solution with various levels of performance and cost, combined into a convenient single solution[5].

Yet, tiered storage solutions are not a novel idea and have existed for many years[6]. What is innovative is the move towards automation of the data movement and the management of the various tiers of performance by making the most out of the data by matching it to its most ideal location within the tiered storage solution.

In recent literature, temperature has been used as a synonym for data priority. It describes current and mission-critical data as hot and less relevant data as cold. This naming convention can also be applied to the storage mediums that contain data. This phenomenon is depicted in Fig. 1, where a small hot tier of storage is displayed at the top. The size of the hot tier is smaller due to the cost of high-performance storage resources. Working down the tiers of Fig. 1, it can be seen that the tiers get larger the further down we proceed. Moving towards cheaper commodity storage results in more storage at each tier; hence, the frozen tiers using cheaper storage hardware are larger.

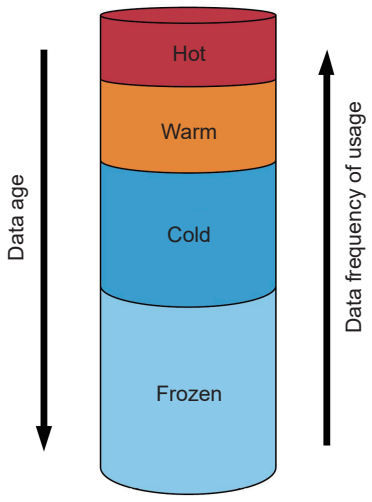


Fig. 1 Typical multi-tiered data temperature storage solution.

The storage of all data within the high-performance tiers of storage is unreasonable and not financially viable. Especially considering that the vast majority of data stored will be accessed only infrequently, if at all. Data temperature is primarily used to describe a dataset's age or frequency of use. These are beneficial factors in determining data positioning in an automated way, but these two factors, used individually or combined, are not enough to position the data in an optimal or time-appropriate fashion.

Many implementations using data age treat new data with the highest priority, and so it gets weighted higher than older data. The problem with this is that just because the data are new, it does not mean that it is required or will get used at all. Age also presents issues when combining with other variables; as data get older, the weighting to move it to colder storage increases. This increase in weighting towards colder storage could conflict with data with a high frequency of use, causing it to rank lower than usage frequency would otherwise determine.

Frequency of usage puts the most popular data in the higher performing resources. The primary problem with this is that the most popular data do not equal the highest priority data. Day-to-day sales processing might result in specific datasets ranking higher in usage, but data used by a project requiring high performance may only get queried once or twice per day. The later data have a much greater need for hot storage, but this is not reflected in the low frequency of usage.

Existing works have no focused consideration for the quality of the data (type and size) or of the storage devices they have moved the data to because they do not factor in user needs or actual real-world priorities (e.g., end-of-year calculations). It is a reactive process only, not a proactive one; a proactive approach would ensure that the data are where they need to be before the queries hit. Also, no consideration has been given to the actual cost of data movement and recursive patterns in the flow of data that climbs up the storage tiers and drops back down. This is a movement that would incur unnecessary input output (I/O) costs.

The purpose of this research work is to address the above-mentioned limitations of existing works. Preliminary research found that there was very little or no information available about the application of

alternative variables in data temperature managed multi-tiered storage solutions. Existing data temperature solution usage of the variables "age" and "usage frequency" is effective. However, in certain scenarios, data placement based on these two variables alone can present a detriment to performance. Such scenarios require the addition of other variables to ensure the continued optimal placement of data.

The proposed data temperature supplementation method is intended to address the challenges posed by the ever-increasing volume of data and its effective management. It is acknowledged that the current classification of data into hot and cold categories based solely on data age and usage frequency is reactive and restrictive. To overcome these limitations and enhance data positioning, the proposed method introduces new variables and conditions, allowing for more intelligent decision-making when identifying hot and cold data. By incorporating additional metadata and user-defined variables, the data temperature value is expanded to encompass a wider range of data attributes. In addition, the proposed rules and conditions optimize data movement, thereby reducing unneeded I/O costs during data transfers between storage tiers. The introduction of a hybrid algorithm that combines existing and new variables improves precision, performance, and user control for optimal data placement within multi-tiered storage solutions, thereby presenting a proactive and efficient approach to data management.

The objective of this research work is to develop additional variables and controls that apply to data temperature, to more accurately determine the optimal positioning of the data within multi-tiered storage solutions, but also to reduce the overall I/O cost of such data movements. The proposed system identifies and tests suitable metadata variables and user-defined variables that can be used to extend data temperature. It establishes rules and conditions for limiting unnecessary movement of data, which will prevent wasted I/O costs. A hybrid algorithm is also proposed that combines existing variables and new variables and conditions into a single data temperature.

This work contributes to the field of multi-tiered storage solutions by proposing an improved approach to data temperature management. Beyond data age and usage frequency, it introduces new variables to more precisely determine optimal data positioning and prioritize data based on multiple factors. By combining these variables into a hybrid algorithm, the proposed system provides greater data placement accuracy and user control, while reducing unnecessary data movement and associated I/O costs. The research addresses the limitations of existing works by providing a proactive approach to data storage, ensuring that data are placed appropriately prior to queries, and taking into account real-world priorities and user requirements. This research aims to improve storage efficiency and performance in environments with multiple storage tiers.

2 Literature Review

This section presents the knowledge and existing work that are being performed to optimize large-scale and multi-tiered storage systems. Recently, researchers have carried out studies to manage clusters consisting of multiple servers by using a single system[7]. They have developed a backend and a front-end portal to manage, dispatch, and rotate jobs and tasks to the backend servers. The jobs and tasks that are of equal priority are handled on a first-come and first-serve basis, with a buffer maintaining the job queue.

Recent advancements in the cloud computing domain also revolutionized the concept of multi-tiered storage systems[8]. Most recent cloud systems by Amazon, Google, Microsoft, and IBM have introduced the concepts of "shared cloud" and "on-demand" resource deployment[9]. Their remote clients are facilitated with resource provisioning on the basis of requirements and demand. Application programming interfaces (APIs) are developed by the providers and used by the clients to achieve on-demand resource deployment.

Most of the recent enterprise multi-tiered storage systems work on a combination of SSDs and HDDs[10]. SSDs have the advantages of speed and performance, but they also have the disadvantages of cost and the number of writes. Therefore, the top tiers of these multi-tiered storage systems use SSDs so that the hot data can be accessed more quickly compared to the lower tiers, and the lower tiers contain HDDs so that the overall storage capacity of the system could be increased.

One of the major challenges faced by such systems is ensuring the diversity of service level agreements (SLAs) in resource provisioning and management[11]. The multi-tiered storage solution also helps tackle this challenge by maintaining the shared resources being

used by different remote clients in multiple tiers.

Another solution that is proposed to handle the challenges of multi-tiered storage systems is to define proper policies based on the usage and frequency of tasks and workload on multiple nodes[12]. The policies are developed in such a way that they handle normal workloads as well as the bursty workloads. Bursty workloads are those that are capable of generating many I/O operations in response to a single query or statement. Such bursty workloads are usually responsible for creating disastrous effects on the overall system. The policies are written in such a way that these workloads are handled by SSDs instead of HDDs.

Numerous algorithmic and theoretical approaches have also been proposed recently by a number of researchers to handle and manage multi-tiered storage systems. These algorithms and theoretical studies tend to observe the effects of data transfer or migration across the system. One of such theories is the edge coloring theory proposed by Ref. [13]. This theory uses polynomial-time approximation to minimize the effects of data transfer across the system and try to achieve the most optimal position.

Another recently proposed technique is an adaptive controller named "triage", which utilizes various ways to mitigate the effects of performance isolation in multi-tiered storage systems. This system is designed to ensure that the distributed environment has high resilience to heavy workloads[14]. This system was later improved by Ref. [15] to tackle the issue of overhead while transferring data from one location to another. This was achieved by identifying the hotspots and tuning the system according to the bandwidth ratio.

Extent-based dynamic tier manager (EDT-DTM) was proposed by Ref. [16] which is a tier management system aimed at dynamically extending the placement during system execution. The proposed tier management system helped reduce the consumption of power by employing dynamic extent placement. Similarly, another attempt was made by Ref. [17] to overcome the issue of disk replacement and transfer of data within a reasonable amount of time. Another algorithm known as the lookahead data transfer algorithm was developed by Ref. [18] to improve the efficiency and performance of multi-tiered storage systems. The idea was to curate a lookahead window size that could help with the needs of dynamic workloads.

However, most of the approaches surveyed in this section assume that the jobs and tasks within the system observe a cyclic pattern, and these approaches do not take into account the different application SLAs in their algorithms. Also, if we look from the service provider's side, these approaches are unable to efficiently manage most of the challenges posed by multi-tiered storage systems, especially in the case of big data. Providing high performance and quality across these hybrid multi-tiered storage resources is still a core and difficult challenge.

We also argue that most of the state-of-the-art studies do not consider both the on-the-fly data transfer issues and a number of application SLAs for data transfer in multi-tiered storage systems.

3 Data Temperature Preliminary

This section presents the knowledge and existing work necessary for understanding the proposed methodology. This section is divided into relevant topic areas that underpin the research into data temperature, with each topic providing a broad understanding before expanding to elements relevant to the proposed problem and solution.

3.1 Tiered storage

Tiered storage was introduced to balance the demand for storage capacity and system performance. The demand was a result of high-performance storage's expensive pricing and exponentially growing data. Tiered storage means placing data across different tiers of storage, with each tier having different levels of availability, performance, security, and reliability, among other considerations. The primary goal of tiered storage is to reduce the cost of ownership[19].

A standard method of deploying tiered storage is that of a three-tiered storage hierarchy model[20]. The model splits storage into tiers, with each tier being assigned a classification of data and the type of storage technology associated with it. These tiers can be defined as follows:

- **Tier 1:** Primary: Mission critical data that support customer-facing and revenue creating operations. Utilize the most expensive, high performance disk systems, to ensure high response, near-zero downtime, and high availability.
- **Tier 2:** Secondary: Broad range of business applications such as databases, file systems, email, and

various other systems. Tier 2 still requires a reasonably fast response time but not as fast as Tier 1, so lower performance disks can be chosen.

● **Tier 3:** Archival: The fastest growing storage tier, used primarily for archiving old data or maintaining data for compliance reasons. This tier uses tape or off-site data vaults. These data are not actively used, so slow responses are acceptable.

Recently, an additional tier of storage often labelled "Tier 0" has emerged, often based on flash based storage. Tier 0 is suited for high values. Tier 0 contains extremely time-sensitive solutions such as financial trading situation where every millisecond counts. This tier would not be expected in day-to-day business requirements and would only occur in niche specialist environments.

Tiered storage was primarily used in specific enterprise environment use cases. However, two challenges limited the success and widespread usage of tiered storage, and that was data movement (migration) and data classification.

3.2 Data classification

Data classification is a data management tool for categorizing data to effectively answer organizational questions about data such as: what data types are present, or who has access to certain types of data. The benefits of classifying data are often data compliance or security related.

Classifying data within tiered storage specifically is the act of matching data and devices to assign data to the most optimal storage tier. It is often the first step when planning a tiered storage environment. Implementations of tiered storage have often been limited due to the challenge of manually classifying data and the lack of effective methods of automated classification.

The following are just some of the variables that have been used as criteria for classifying data, they have been split into three logical groupings:

● Data metadata
– Data type,
– Assigned owner,
– Location,
– Name.
● Data time
– Creation time,
– Last accessed,
– Last update time.
● Data content
– Specific tags and details about the sort of data that are actually stored (financial, sales, etc.),
– Private or publicly accessible,
– Availability requirement/impact of unavailability.

3.3 Data migration

The majority of movement policies are mono-directional data movement only occurring from high performance tier to low performance tiers, without much consideration for bidirectional movement of data. The earliest research into tiered storage was focused solely on the movement of data out to tertiary (offline) storage, decluttering storage and freeing up higher performance resources for new incoming data[21].

3.4 Multi-temperature storage and data temperature

Multi-temperature data storage is a tiered storage strategy aimed at minimizing the costs associated with maintaining large amounts of data. Data are assigned a "temperature" based on age and frequency of usage, then the more frequently accessed or newer data are optimized for placement into "hot" high-performance tiers of storage, with the "cold" data being isolated and placed into cheaper commodity storage solutions. The assignment of temperature and movement of data in a multi-temperature storage environment is an automated process.

Interest and implementations of data temperature have appeared in a lot of enterprise data warehouse environments with big names such as IBM, Teradata, and SAP all having their own implementations. Apache Hadoop distributed file system (HDFS) even has storage types and policies designed around data temperature as part of an archival storage solution, the goal being to decouple ever-increasing storage requirements from compute capacity[22].

3.5 Data usage frequency

Data usage frequency, when used as a value for data temperature, is the identification of the amount of times data are accessed, the higher the number of times the data have been accessed, then the hotter the data's temperature. In general, this is great and makes perfect sense, the most critical data are the data that more people are requesting, so placing that data item in a high-performance storage area will be the most beneficial, serving the needs of more users with much

higher response times.

The problem with this method of thinking is that it only works if all data and queries made against the system have the same level of importance and urgency. Being the most required does not immediately equate to the most important. Figure 2 depicts two data items in a data temperature managed storage environment using frequency of use as its variable for data temperature. The figures multiple general usage users are querying a data item thousands of times per day, and so within the usage-based data placement, it is allocated a prime position within the hottest storage tier. In comparison, the single mission-critical single user that requires responsive query times to work efficiently is instead working on data within the lowest performance "frozen" tier due to the limited usage of the data item.

It would be more beneficial to upgrade the mission-critical data item to the hottest tier and downgrade the more generic data item to a warmer tier of storage, but within a usage frequency controlled environment, then this could not happen. The example presented as part of Fig. 2 could also raise issues with I/O, the "hot" tier of storage is most likely a single or small group of high performance nodes, whereas the cooler tiers of storage are going to most definitely be much larger clusters of nodes that would better serve thousands of queries from multiple users, leaving the high performance nodes free to a limited but high priority set of users and datasets.

Another issue that could prove costly in a usage-controlled data temperature is that of recurring movements of data, by this, we mean that data are not always used once and then never used again. Instead, some datasets are used a lot for a period, and then the usage drops off, but then consider that the period of usage repeats repeatedly, for example once every month. On first thought, this is ideal for the frequency of usage and could be its prime sort of problem when it resolves.

The problem although is that it is not a simple act of these data, which was cold but is now hot; it is a gradual process as the usage of the data ramps up, so the data could potentially be climbing up the storage tiers as the usage increases for that month, using the storage solution from Fig. 2 as an example that could be three movements of data upwards, then three movements back down the tiers when the period of high usage ends, this wasted data movement demonstrated in Fig. 3a, ideally because of the consistent movement every month instead only two movements of data as demonstrated by Fig. 3b could have sufficed, this would result in a lot less wastage of resources every single month.

It would also result in users benefiting immediately from him high-performance tier, instead of the wait for it climbing up the storage tiers. It is also entirely feasible by the time the data had climbed the tiers of storage, the period of usage could be over for that month, and so any benefit the "hot" high-performance tier of storage provided was wasted.

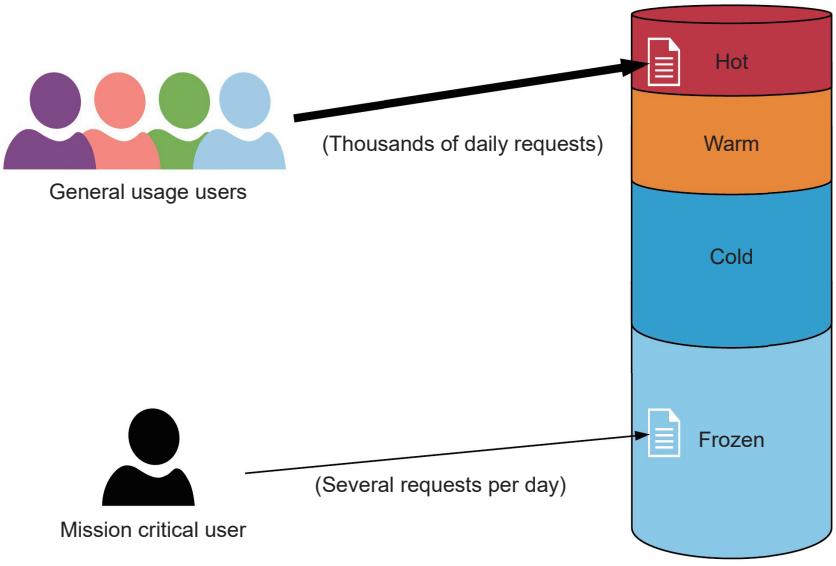


Fig. 2 Usage-based data placement.

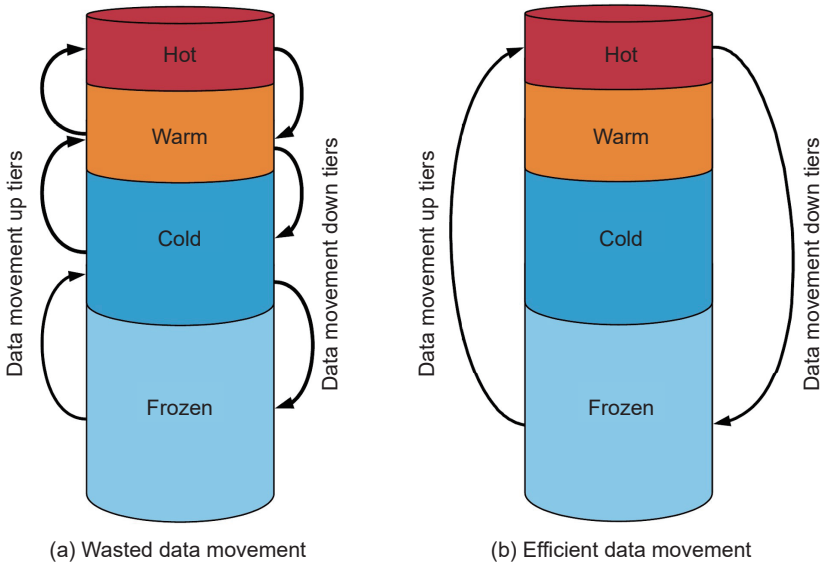


Fig. 3 Wasted data movement diagram.

3.6 Data age

Age is commonly applied in one of two ways, the first being that of the newest data entering the system translating directly to the hottest data and placed in a high-performance storage tier. Second, data can instead be allocated a default temperature, commonly not hot nor cold, but an average temperature value and more suited to a commodity storage location. Both feature a decrease in temperature over time as they get older.

The problem with data age as a value for determining the hotness of data is that it automatically assumes that new data have a level of importance and that old data are less important than all incoming new data. For specific use cases, this can be ideal such as eBay's usage of data temperature, where they clearly identified that a common trait of their datasets is heavy initial usage, with new datasets being considered hot and decreasing in how frequently they were accessed over time.

This sort of consistent pattern of data being input and then used immediately is not typical of all use cases, instead consider, for example, a pharmaceutical company where new datasets are ingested frequently from various labs and subjects but the actual data being used most frequently is the data for active studies and projects and not just the newest available sets of data.

In the use-cases described above, where data are not consistently queried immediately upon ingestion, we are presented with the issue of data no longer being in high-performance storage when it is needed. Figure 4

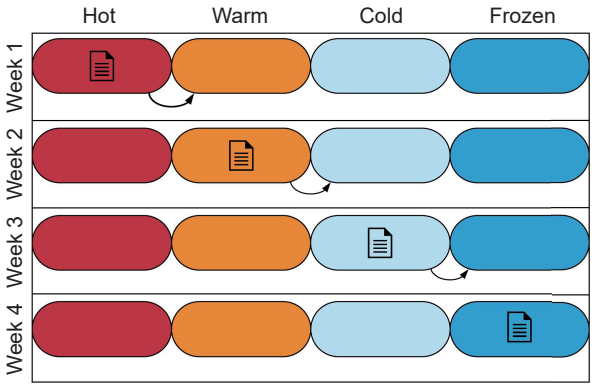


Fig. 4 Data ageing to lower tiers over time.

demonstrates this point, in Week 1, the data have been inserted straight into the "hottest" high-performance storage area. The problem with this is that the dataset is not going to be needed for several weeks, but over the four-week period presented in the diagram, the data have gradually been migrated down each week to achieve a "colder" lower performance storage tier because they are now old data. The result of this sort of movement is that the storage tiers are not well utilized, as the data that currently need querying are instead in the lowest performance tier of storage and will be much slower to work with than they were four weeks prior.

Another issue with data age is that today's volume of data are enormous, so the amount of new data entering a storage solution will always push older data down to lower tiers in favor of the newer data, even if older data were only ingested into the hot tier seconds prior.

Figure 5 depicts this issue over a 20-min period across four storage tiers: hot, warm, cold, and frozen. Data are represented by circles, with new data inserted over time, and so the newer data push the old data down into the lower storage tiers as new data objects get ingested.

The scale of the example in Fig. 5 is vastly minimized to use 26 alpha characters, but considering today's volume of data created, it is not that unrealistic for GB's or TB's of data to be ingested over the 20-min period presented. Taking note of data object "M", after 20 min and two ingestions of new data, it has been degraded from data with "hot" performance and is instead residing in storage tiers three levels down. The downgrade is due to limitations in space on the high-performance tiers, requiring the "M" data item to be gradually migrated between tiers to accommodate the newer "hotter" data.

3.7 Lack of user control

Data temperature as a data resource management solution is mostly an automated process, with age and usage frequently being values that are easily tracked and acted upon by the system, without any prior user input. The problem with automation of data temperature is that it only has two points of data to work with: age and usage, but this does not account for the real world or business requirements; instead, all data are treated as if it were all the same. In many instances, this equality of data is acceptable, but for use cases where specific data do take priority, that data need to be in the "hot" high-performance storage tier. This will help to attain maximum benefit of the high-performance tier.

Despite this section arguing against the total lack of user input and presenting it as an issue, too much user input would also cause its own specific set of problems, not even counting the person hours and resources for manually allocating data across a tiered storage solution. Any solution proposed should instead allow for greater user control in a more supervisory manner and not a hands-on manual process, as this would prove to be a regressive and not a progressive action towards smarter automation of a tiered storage solution.

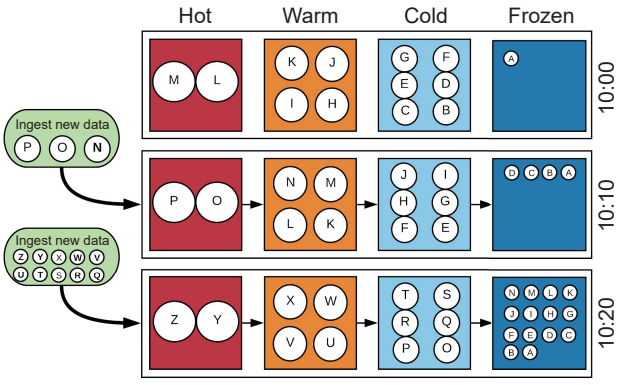


Fig. 5 New data pushing older data down to colder storage tiers.

The discussion in this section concludes that in the past, we have had tiered storage, but due to issues of classification and data movement, it never really took off in any real way except within specific enterprise environments. With a resurgence in tiered storage in the form of data temperature, prompted by lowered costs of memory, SSDs and various new storage devices that have varying levels of cost, this, combined with virtualized storage and file systems such as HDFS, makes migration of data feasible due to its lower impact on system performance.

Data temperature implementations currently only classify data into two variables that, as discussed above, have various flaws inherent to each, at least outside of specific use cases. Older tiered storage data classification was costly and inconvenient, but specific traits of the data itself or users were explored and used. Caching, a technique with similar characteristics to tiered storage, also has a method of classifying data. The traits of both expanded beyond the age and usage currently used for data temperature.

Metadata and predictive algorithms have come a long way in recent years, simplifying the classification of data and also allowing for pre-emptive predictions of how to classify data in the future. From this research, we have concluded that data temperature classification can expand by exploring existing variables previously applied in tiered or cached environments. This, combined with complete metadata management tools and machine learning algorithms against the data, provides multiple data points to classify the data and optimally place temperatures at the ideal temperature and location.

4 Extending Data Temperature

Existing work revealed that the variables that have been used to determine the placement of data in the cache were merely age and usage. From this, we propose the following: the utilization of alternative

variables, but also a process of combining multiple variables into scope-specific configurations to deliver a situational based solution that proved to be more optimal than the current single solution approach. We also propose an approach to balance the temperature of our temperature-ordered data with the goal of reducing unnecessary movement.

We propose that instead of being constrained to two variables that are of limited scope in how we rank data in a multi-temperature system, considerations should instead be made for alternative variables when ranking data and assigning a "temperature". These variables are then further combined with conditional statements using solution-specific ranking algorithms to provide optimal placement and greater user control of data.

The basic premise of the proposed solution is already prevalent in existing data temperature implementations, with data age and usage frequency often being combined and used together to deliver a more competent data temperature value. The deficiencies of age and usage as variables for optimally positioning data were explored in the prior section, but when combined into a single data temperature value, these identified deficiencies are mitigated. When combined into a single data temperature value, age gradually lowers the temperature over time to slowly decay the data into lower tiers. When combined with the frequency of use, it prevents data still actively used from getting archived.

The use of more data points often brings greater understanding to a variety of problems, but even when combined, data age and usage frequency have the issue of being of limited scope and only being relevant when the newest and most highly used data reflect importance; it cannot consider operational or specific data requirements, and instead all data are generic and treated equally. The scope of the current data temperature does match a lot of use cases, such as those where you have lots of new incoming data that are of immediate importance and will be of immediate use, but not all requirements are so simplistic, so a demand exists for more tailored case-specific data temperatures that will provide a more optimal placement of data on a case-by-case basis.

The proposal of additional variables and controls over data comes from the process of data classification that has been applied to tiered storage for many years. Data classification has long been used in many variables such as owner, filename, time last accessed, business value, availability index, retention period, etc. For determining how data should migrate between tiers of storage, caching has also been explored to identify what should exist in the cache and what should not, with similar usage of a broader range of variables than just age and usage alone.

The proposed approach intends to introduce additional variables that, when combined, further mitigate the deficiencies of using a single data temperature variable, so the existing variables of age and usage frequency are not being replaced as there is nothing explicitly wrong with them. They just fit a specific use case and are not the correct tool for every situation; we are instead adding additional variables to deliver a more flexible toolset when addressing data temperature in a variety of usage scenarios.

We have split the process into five stages; the first three are the initial setup and configuration of the system that get performed when establishing what data temperature means within a specific system. The final two stages are processes that repeat continually as the data temperature of a system is calculated over time. The frequency of repetition will be domain specific and influenced by various factors such as the amount or regularity of new data added, the amount of active system usage, and many other factors, but for the scope of our research and within our experiments, we have worked under the assumption that updates to data temperature occur overnight every night. The proposed data temperature process consists of the following stages:

- Identify target system scope,
- Selecting suitable variables,
- Applying conditional rules,
- Calculating the temperature of data,
- Balancing data temperature.

The following subsections cover each stage of the proposed process. Each stage is thoroughly explained and justified before concluding and moving onto the next stage of the process.

4.1 Identifying target system scope

The proposed approach is not a recommendation that all the proposed new variables be used together; instead, consider it a technique where a subset of relevant variables is identified to construct a model that is more accurate and suitable to a specific use case than if the entirety of an available dataset is used instead.

Identifying the scope of a storage system consists of fact-finding tasks that will aid in the selection of these relevant variables. The aim here is to identify how to use the system, and how the "hotness" or "coldness" of the data should be determined using the variables we have defined. System scope can be determined by exploring (but not limited to) the following conditions:

- **Users requirements:** Not all users are equal, access rights, responsibilities, and job roles have a significant impact on how a user interacts with a system. Having understood that a specific group of users requires data available with minimal delay, it allows that data to have a higher priority than data required by more generic users.
- **Objectives of the system:** The purpose of the system and what it is meant to achieve influence how we decide what data should move through a system, there is a big difference between a system used specifically for analytics and a system that is intended for archiving purposes only.
- **Additional interfaces and influences:** Storage solutions are not always user-centric. Often, they can instead be used by specific tools, such as machine learning or various other automated processes. The problem with this is that the query patterns and usage of the system would be much different compared to what you would find if compared to a set of users, potentially resulting in less focus placed on these tools.
- **Frequency of updates:** Daily updates equal a lot of movement around a system, especially if the new data are essential and need prioritizing and so become an important factor, less frequent update is a much slower and less movement intensive scenario, so in the former scenario, identifying variables that decrease movement would be ideal.
- **Concurrent users:** The number of users is not indicative of what is important, but it does allow us to understand the weighting of variables such as read or write frequency and make more knowledgeable decisions on how to place data.

Without first taking such steps to understand a storage solution, there is little benefit to be gained, and instead, the existing data temperature implementations could be used and applied to everything regardless of any potential loss in performance due to incompatibility with the requirements of the system. Understanding the storage system allows for a movement towards a more optimal storage solution, not the most generic optimal storage solution, but a storage system that is most optimal for a specific use case scenario with a bespoke data temperature. In understanding a system, we can identify appropriate variables that match the scope and requirements of the system more appropriately.

4.2 Selecting suitable variables

We propose the following variables that can be used as new temperature variables. These variables include "file name", "file type", "tagging/categories", "who created", "when created", "who has access", "read frequency", "write frequency", "movement count", and "movement direction". The proposed variables are either simple metadata such as names, tags, etc., or can also be statistics of the data, such as the amount it moves in a month or total write counts. This is not an exhaustive list of all potential variables; instead, it is a brief selection of variables that could be used to contribute towards our proposed data temperature value, with each item accompanied by an example use-case for determining hotness or coldness.

The selection of the variables will be determined by the initial evaluation performed on the system to determine its scope; at present, we propose no specific process for automatically classifying or determining the relevance or suitability of a specific variable for a specific system or use case. Variable selection is instead a process that will require domain and system knowledge.

4.3 Applying conditional rules

The secondary aim of the proposed work is to introduce greater user control over how the data are placed, even with complex machine learning and continuously evaluating years worth of usage data. Users will often have specific knowledge external to the system that could ensure optimal placement of data for a specific event, such as an urgent project where specific data could experience a surge in usage for only a single day.

No system could have expected that, but with the addition of a conditional rule indicating that for that day a specific data type would experience heavy levels of usage, it could already be in place (within hot storage tiers) ready to be used and delivering the optimum performance available. Conditional rules are not proposed just with user input in mind, but could also be used to input events and conditions that have

been recommended by a machine learning algorithm. Examples of conditional rules are as follows:

- Data age ⩾ 5 months = archive.
- If current day is Monday, all PDF files are high priority.
- Filename containing AUG2016 = hot data.
- If write frequency ⩾ read frequency = hot or warm.

Similar to the choice of variables, we make no specific recommendations on which conditional rules to apply. These rules are case specific, so what would be applicable to one storage solution would not be applicable to another. The rule statements used above allow for broad classifications of data, but can also be made very specific if required. These rules make use of the identified variables and give them more depth and specificity. All are determined by the user and providing a significant amount of influence over how we store the data.

This proposal for conditional rules is that they can be a constant fixture of the data temperature calculations, or that the rules can be applied as needed at a moment of notice. This flexibility in customization allows for even greater responsiveness of the data and its placement and movement throughout the system.

Conditional rules will be applied exactly the same as standalone variables and combined into a single data temperature score, with the option of determining a binary outcome or a ranged outcome as per the number of available tiers.

4.4 Data temperature scoring algorithm

The primary objective of a hot and cold storage solution is to distinguish between frequently accessed or "hot" data and rarely accessed or "cold" data based on their relative importance. The optimal scenario entails sorting all data from the hottest to coldest, ensuring that critical data are stored in high-performance storage and less important data are stored in lower-cost, slower tiers. However, the complexity of sorting increases when multiple variables are considered, necessitating multiple comparison sorts for each variable. A data temperature scoring algorithm is proposed to address this issue. This algorithm applies scaling and weighting mechanisms to each variable and rule, generating a single numeric score for each data object that represents its "hotness". The algorithm employs a more efficient counting sort to organise data objects based on their calculated scores, optimising the storage solution by placing data in the appropriate tiers in an efficient manner.

The proposed data temperature scoring algorithm streamlines the sorting of data based on multiple variables in hot and cold storage solutions. Combining scaling and weighting mechanisms, it computes a single numeric score for each data object that reflects its relative significance. This eliminates the need for multiple comparison sorts per variable and replaces them with a more efficient counting sort, enabling the storage system to sort data from the hottest to coldest. The capability of the algorithm to optimise storage performance ensures that frequently accessed data are readily available in high-performance tiers, while less frequently accessed data are stored in lower-cost, lower-performance tiers, resulting in a cost-effective and efficient multi-tiered storage solution.

Figure 6 depicts the scoring process per data object, at the top of the diagram, the variables and conditional rules that are applied to each data object are being inserted into the process, with the final output of the process being an individual score per each data object. The phases of the process are explained in detail in the subsections below.

4.4.1 Scoring per variable/conditions

Scoring is a single process that is applied to each variable of a data object, but due to the multiple variables and conditionals used, we have split the process into binary or ranged scoring. Binary scoring is applicable to variables and conditions that only have a binary outcome, and ranged scoring is instead for outcomes that can be split into three or more outcomes, the number of outcomes is determined by the number of available tiers.

Each variable/conditional that has been identified for usage is scored equally, meaning the maximum or minimum score available is equal for each variable/conditional (not considering the addition of weightings in the next stage). The maximum score range is based on the number of storage tiers in usage, so 4 tiers of storage result in a score range of 1 through 4. Within the topic of feature scaling/normalization, this would be similar to the min-max scaling approach. Min-max scaling is an alternative approach to Z-score normalization. Min-max has a bounded range and so suppresses the effect of outliers with its fewer initial standard deviations[23].

Range scoring is applied to variables/conditions that

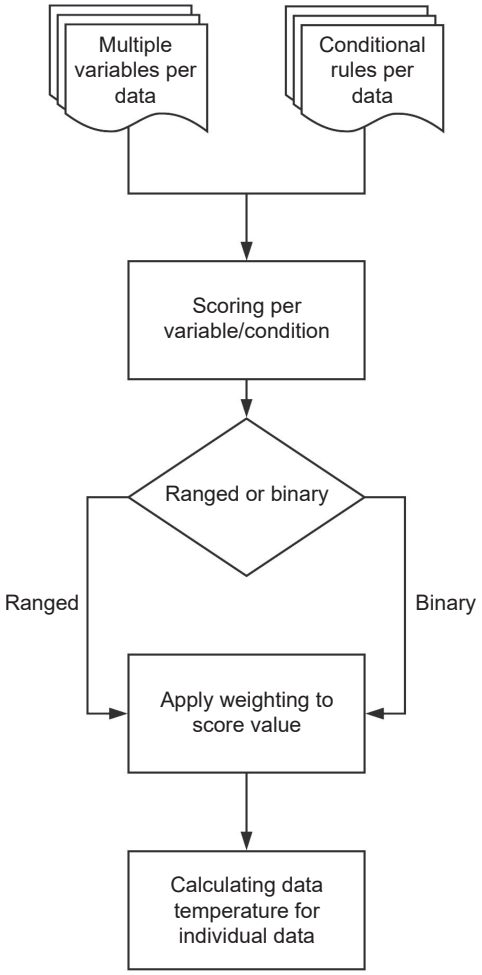


Fig. 6 Overview of the data temperature scoring process per variable/condition.

can be split up into multiple ranges, the number of ranges that are created is based upon the number of tiers of storage available. An example of this is presented in Table 1, which displays the scoring range of a 4-tiered storage for the variable of data age. The ranges themselves are currently created manually, but these ranges can be generated automatically according to the suitability of the storage tiers.

In situations where a variety of variables were impractical or impracticable, binary scoring was implemented as an alternative method. In situations where a range of values for a particular variable cannot be determined, the binary scoring system provides a simple and effective solution. This method of scoring uses the maximum and minimum values of the range scoring system to represent "true" and "false" conditions, respectively. In the binary scoring method, a conditional rule is defined to determine whether a particular data characteristic is present or absent. For instance, the passage describes a conditional rule that seeks the presence of a CSV data type. If a data object is determined to be of type CSV, it is assigned the maximum possible score (for example, 4) to indicate that it is of greater importance (hot data). Alternatively, if the data object is not of type CSV, it is regarded as having a lower priority (cold data) and is assigned the lowest possible score (for example, 1). Table 2 illustrates this with an example scenario. Table 2 displays various data objects and their respective scores according to the binary scoring system. If the data type is CSV, the data item is given a score of 4, indicating that it is a hot data item. If the data type is not CSV, it is assigned a score of 1, indicating that it has a lower priority in the storage system and is therefore less important.

Binary scoring is a straightforward and efficient method for addressing situations where a simple true/false determination is adequate. It complements the more complex range scoring system and expands the data temperature scoring algorithm's adaptability to accommodate a variety of scenarios in multi-tiered storage solutions. By incorporating binary scoring in addition to range scoring, the algorithm becomes more flexible and adaptable to a wider range of data characteristics, allowing for improved optimisation and decision-making in data placement within the storage tiers.

Another benefit of using these ranges is that the temperature score is not weighted specifically to one data variable, it is instead relatively equal due to the normalization and balancing that occur. This also means that the addition of user specified rules does not totally dominate the temperature; instead, they are factored together equally. This is in part to prevent user

Table 1 Example of a range score for a 4-tiered storage environment.

Scoring range	Score description	Score value
Data age < 1 week	Hot	4
1 week ⩽Data age < 2 weeks	Warm	3
2 weeks ⩽Data age ⩽ 1 month	Cold	2
Data age > 1 month	Frozen	1

Table 2 Example of a binary score for a 4-tiered storage environment.

Data type	Score description	Score value
If data type is CSV	Hot	4
If data type is not CSV	Frozen	1

requests from totally dominating the score and let the combined score overall determines where the data are actually positioned.

Considering our proposed example of 4 tiers, if we assume that each data item temperature score is calculated by 6 variables (combination of ranged or binary), the maximum total score is going to be 24 as can be seen in Fig. 7. Similarly, so at the minimum, the minimum coldest score available would be a score of 6 as can be seen in Fig. 8. If we increase the tiers or variables used, the min/max scores will also increase, but the opportunity for each data object to score the same remains balanced.

4.4.2 Weighting

Here, we determine the relative importance of the variables identified within the scope of our system. The assignment of weight to a specific variable is largely dependent on the scope and evaluation of the specific system. The weighting can pertain to multiple factors such as user benefit (all users or specific users), operational requirement (regularization needs, etc.), I/O performance, archiving procedures, update frequencies, or data size.

Due to the scale of data available, instead of ranking all data variables in comparison to each other, we instead gather the values from our identified variables, apply the weighting identified above, and then combine them into a convenient score per data item; this score represents the temperature of that specific piece of data.

The standard weighting of each variable or conditional is 1, with an increase or decrease in this weight as needed to increase or decrease the priority of the specific value; e.g., a weighting of 0.5 would reduce the impact of that specific score, but similarly, a weighting of 4 would greatly increase the amount of impact a single variable has upon the overall temperature.

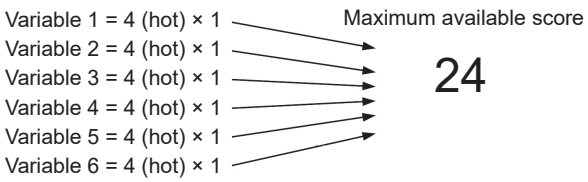


Fig. 7 Scoring per variable, all hot scores.

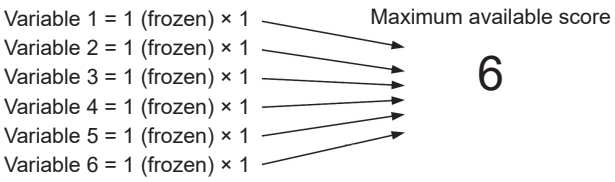


Fig. 8 Scoring per variable, all frozen scores.

The scaling/balance afforded by the prior section should be adequate without the need for additional weighting, but for added user control, we propose the addition of applying a weighting to each individual variable of a data object.

We propose a minimum range of 0.5 and a maximum of 2. This is to prevent user bias from heavily influencing the scores while still allowing for customization within the existing variables used. Using the examples in the prior section as a basis, the potential lowest score available now becomes 3 and the highest available score becomes 48, as presented in Fig. 9.

The min and max above are not realistic, as if you are applying a max or min weighting to everything, then you might as well apply it to nothing. A more realistic example would be applying it to a single variable such as in Fig. 10. The double weighting applied to variable 1 means that it will overall score a higher value compared to the other variables, even if it scores 1 from a colder tier, it will be weighted higher and more important than 1 from the other variables.

4.5 Calculating the temperature of data

The final process of calculating each data object's

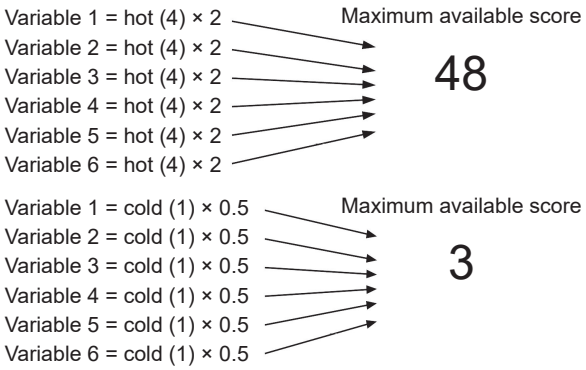


Fig. 9 Scoring per variable, but with double or half weighting applied.

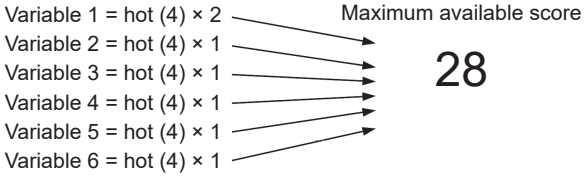


Fig. 10 Scoring per variable, but with max weighting applied to a single variable.

individual temperature value is that of combining all the scores and weights of each individual variable and conditional rules into a single data temperature. Due to the normalization/scaling of the data objects, we can simply total the scores from each variable used together and have an independent data temperature score for each data object within our system.

The proposed calculation is found within Algorithm 1, it loops through all variables and conditional rules that were applied to the system, first getting normalized/scaled by either the ranged-outcome or the binary-outcome methods dependant solely on the type of value of the variable used. Once a normalized score has been returned, it is then multiplied by that variables assigned weighting to increase or decrease the priority of that variable as is needed. Finally, Algorithm 1 just totals up the score achieved for each variable and the conditional rule, dataTemperatureScore being the data temperature value that is assigned to each data object within the system.

The benefit of an independent score is that it never increases in complexity or resources required regardless of the amount of data actually stored, no sorting or reliance on thousands or millions of other data items and how they score.

Algorithm 1 runs against each data item and makes all assigned variables and conditional rules that have been identified prior, the final outcome being that of a single data temperature value created per data item. The output score is a singular value per data item, the score is stored alongside the metadata about each data object. Algorithm 1 itself is low complexity due to the normalization and scaling that occurred per each variable/rule, without that process, the majority of Algorithm 1 would instead be aimed at equalizing the different values and achieving some semblance of balance, so specific values were not unfairly weighted.

Algorithm 1 Calculating overall data temperature of an object

```
Data: variables, minScore, and maxScore
Result: dataTemperatureScore
foreach var in variables do
    if var.type == "range" then
        if var.value == Condition 1 then
            var.score = maxScore;
        else if var.value == Condition 2 then
            var.score = maxScore(−1);
        else if var.value == Condition 3 then
            var.score = maxScore(−2);
        else if var.value == Condition 4 then
            var.score = minScore;
    end
    else if var.type == binary then
        if var.value == true then
            var.score = maxScore;
        else if var.value == false then
            var.score = minScore;
    dataTemperatureScore = var.score × var.weight;
end
```

Dependent on the variables and other conditions such as update frequency, the output value could change greatly between each run, but a change in the assigned score does not necessarily equal the movement up or down a tier as each value is updated accordingly and compared against the rest to determine an ideal storage tier in the section below.

4.6 Sorting

Sorting is a comparative action between two values, determining if one value is higher or lower than the other, then applying this across a much broader range to gradually sort these values into order. The problem with this comparative style of ordering is that it has a lower bound of $O(n\log n)$ complexity. $O(n\log n)$ is not the worst outcome for a sorting algorithm, but having to run it multiple times against each of our data object variables and conditional statements would result in a lot of compute and memory usage depending on the scale of data objects stored.

Fortunately, we scaled and weighted our values and provided a single score per data object; we could just sort this score using a comparative sort as described above. However, we have put ourselves in a unique position to use an integer-based sort due to the scaling process to assign every data object a single data temperature score. The specific integer-based sort proposed is the bucket sort.

The bucket sort on average has a complexity of $O(n+k)$, which is a decent linear sort (where n is the number of data objects and k is the number of distinct values); the problem is that the basis for this requires an even distribution between buckets. If one bucket instead contained all values, then we had to sort them into a single bucket, this could result in the worst complexity of $O(n^2)$. By adequately defining variables and conditions appropriate to the storage data, the

worst case scenario should be avoided, and the result will be a more even distribution of data objects across the buckets.

Simplified, the bucket sort algorithm operates as per the following instructions, additionally conveyed in Fig. 11.

- Establish buckets of evenly distributed ranges.
- Distribute each among the bins.
- Sorting occurs within each individual bucket.
- Buckets are visited in order and the elements are output back into the array.

4.7 Balancing data across storage tiers

Once the scoring of data temperature has been achieved and an order for all data objects has been established, the final step of the data temperature process is implemented. The score is used as a comparison against each instance of data to determine whether it should be migrated upwards or downwards to a different tier of storage. The goal of balancing data temperature is not to orderly arrange it from the highest scoring to the lowest scoring; it is instead aimed at migrating between various tiers of storage.

Consider if a specific numerical order of data temperature was identified for every piece of data, and then we cleared out all storage and inserted data into it,

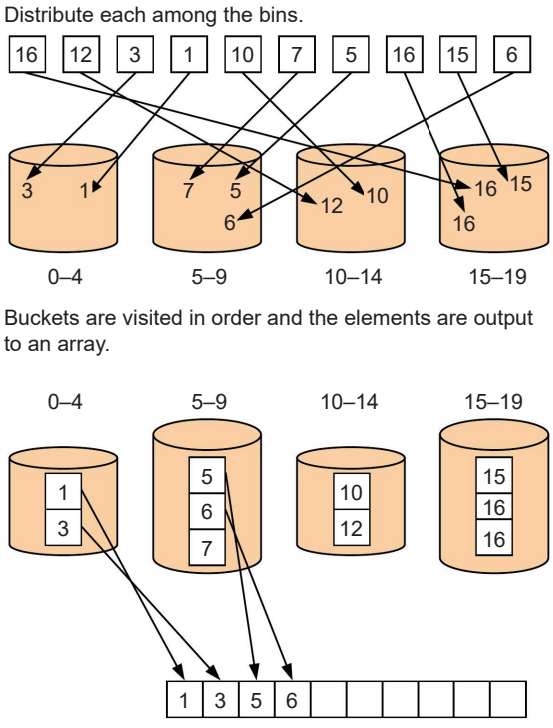


Fig. 11 Diagram depicting a simplified interpretation of bucket sort.

the hottest first, until each tier filled up or the data ran out of the amount of movement required in one instance to actually ensure that all data would be in its assigned position, from the hottest to coldest. This is just wasteful. Instead, consider only moving specific data objects up and down as needed, the data only need to end up in a relevant tier of storage; it does not need to be assigned to the most perfect position within storage.

In an ideal world, the simplistic solution would be to order the data by the generated score from the highest to lowest (the highest scoring being ingested first and gradually filling up each tier until you ran out of storage or items to store). The problem with doing it that way is that it would require a huge amount of movement with potentially every data object being moved, this would be an unnecessary waste of I/O and system resources.

Thankfully, we do not need as much movement as alluded to above; the first item in the hot tier storage will have no difference performance wise compared to the thousand items within the same storage tier. Movement between three or more tiers of storage is a lot more manageable than a complete ascending sorting of every data item, which is unrealistic.

Each of our tiers of storage has a temperature range attributed to it; these ranges are determined by the data present in each tier. For each tier, the average/mean value is stored, and the min and max values are currently stored within each tier. Maintaining these values lets us know how much needs to be scored to move a temperature upwards or downwards between tiers.

Similar to the scoring process itself, the movement occurs per item; the movement process is kept separate from the temperature scoring and occurs after the process completes, to prevent unnecessary movement. Consider that if you calculated a new score for each item and then made the movement action, the next data item score could override that, meaning you would have to move the same data around again.

The premise of this movement is to look at where the data are currently located, only move it when absolutely necessary. The above movement itself does not occur automatically after the above process; instead, it is added to a movement action list. This step has been added to further ensure that data are moved as little as possible. By adding it to such an action list, we

can order it appropriately, working from the highest to the lowest, moving down the tiers as we go.

We have split our proposed approach to balancing data into the following three stages:

- Establishing upper and lower bounds of storage tiers.
- Assigning movement to data objects.
- Data relocation plan.

4.7.1 Optimal data object for tier placement

Before any movement can occur, we need to understand where a data object should belong, but we only have a limited amount of space in the higher tiers, so we need to establish what data will fit in the hottest tiers as well as consider its data temperature score. Thankfully, the counting sort we implemented can help us with this type of calculation. By reusing the object counts from the highest to the lowest, we can easily establish the number of items that will fit into each storage tier.

For example, in Fig. 12, we have a table that will act as a visual description of the data sorted into groups for counting as part of the counting sort algorithm. Along the bottom, there are coloured ranges that depict the span of data temperature scores which the individual tiers actually cover. Note that the value within each range is the amount of storage used and the maximum amount of storage for that tier. Working from right to left (the highest to lowest), we are gradually filling up each tier until we run out of storage for that tier, when one tier of storage runs out the next tier of storage starts counting along (note the coloured background of the cells that make this more visible).

The pseudocode presented in Algorithm 2 details this process: initially, we loop through the collection of our data objects; this collection comes from the prior stage of calculating data temperature, so each data object has

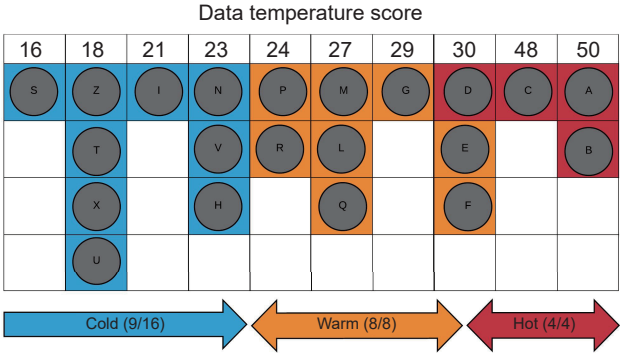


Fig. 12 Filling up the storage tiers capacity from the hottest to coldest.

Algorithm 2 Data object to tier placement

```
Data: dataObjectList, tierList, currentTier, and storageTier tier = 0;
foreach each dataObject in dataObjectList do
    if tierList(tier).tierStorageUsed + dataObject.size > tierList(tier).tierCapacity then
        dataObject into tierList(tier+1).dataObjects; x = tierList(tier+1).tierStorageUsed + dataObject.size;
        tierList(tier+1).tierStorageUsed = x;
    end
    else
        Insert dataObject into tierList(tier).dataObjects; x = tierList(tier).tierStorageUsed + dataObject.size;
        tierList(tier).tierStorageUsed = x;
        if storageTier ⩾ 90 of tierList(tier).tierCapacity then
            tierList(tier).isFull = true; tier = tier + 1;
    dataTemperatureScore = var.score × var.weight;
end
foreach each dataObject in dataObjectList do
    Insert storageTier.dataObjects into storageTier;
end
```

some generic metadata details such as identifier's size, current tier location, and also its data temperature score. The prior process also sorted them, so they are in order from the highest to lowest data temperature score.

For each tier of storage, we have a tier object collected together as tierList in Algorithm 2. Each tier object has an empty collection waiting to be filled by data objects. Looping through each data object, the goal is to insert them into a specific tier. The tiers are arranged from the highest to lowest (the first tier at 0 is the highest tier). This is also the reason it is important that our temperature-scored data be ordered from the highest to lowest, so the more high-priority data are first inserted into each tier. Before this insert occurs, we need to check whether the data object will fit into the tier.

The if statement in Algorithm 2 calculates the amount of space currently used by the tier plus the current object size; if this is above the storage capacity, the data object gets placed into the next tier of storage, and we move onto the next data object in order. The else statement occurs if enough space is available within the current tier for our data object. First, the data object has its proposed tier value updated to match the tier it is being inserted into, then it is inserted into the current tier. Next, the current tier's tierStorageUsed value is updated by adding the current data object's

size to its own.

Finally, a check occurs in the form of another if statement in Algorithm 2. It checks the amount of storage space left, and if it is a certain percentage value full, then the current tier's isFull value is set to true, and we move onto the next tier, this check is done so that we are not trying to insert every value into the first tier if a tiny amount of space is left, but it also acts as a buffer for each storage tier. Once the initial loop over all available data objects is completed, the tiers of storage are looped through and their collections of data objects are combined into a single collection of data objects in the order of the proposed tier of storage, ready for the next phase.

Once the process has worked its way through our data structure from the highest to lowest and split the data objects into specific tiers of storage, we will know what data object's should ideally be placed in which tiers (based upon our data temperature scoring). Figure 13 depicts the information we should now understand about our data. We know the ideal tier of storage for each data object and the data temperature score of each temperature. With an understanding of where our data should optimally be located and an understanding of what sort of score range each storage tier should adhere to, we can now make moves towards planning to move data into the correct tiers of storage.

4.7.2 Assigning movement to data objects

In the prior section, we established where data should optimally be located, now we want to avoid movement. In an ideal world, you could remove all data and then reinsert it at the relevant tier into an empty storage environment, but this is not viable. Instead, we need to assess and plan to move data or not move data as needed.

Working from the highest to the lowest scoring again, we check if the tier assigned matches the current tier, and this informs us if an action needs to be taken. From this, we can draw three conclusions: the first and second are moving the data object up and down tiers, and the third is taking no action at all. This is presented in Fig. 14, with this decision being actioned on every data object.

If the first or second options are the result, then an entry is added to the action list, such as "move data object Y from cold tier to hot tier". This repeats for every data item until we have a full list of movement actions that need to be executed.

Algorithm 3 is the simple process of how we establish what needs moving using the prior processes of scoring, sorting, and establishing an ideal position in storage. With the collection of objects from the prior process of assigning a proposed tier to each data object, here we check that against the data objects current tier of storage. If it matches the proposed location, then nothing happens; the data's location does not need to be changed, but if the tier is different, then a movement action is generated, consenting the data ID and tier that it is to be moved to, and this occurs for each data object.

4.7.3 Data relocation plan

In the past two subsections, we presented what ideal tiers each object should be assigned to and proposed a list of actions that are only going to move what is necessary to move. Movement occurs in reverse, from the lowest to highest, so the higher tiers of storage are freed up, when it comes to moving data up. If all movement down to the lower tiers has already been completed, then there will be no conflict with space requirements when trying to move data upward to the higher tiers, where space is more limited. The proposed solution may become problematic if even the lowest storage tier is of low capacity, but in most cases, we can think of the lowest commodity tiers of storage as being almost unlimited at this point, at least in a majority of cases.

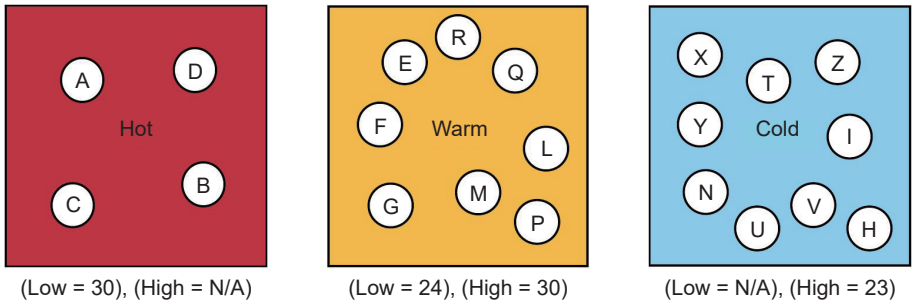


Fig. 13 Optimal tier data object locations.

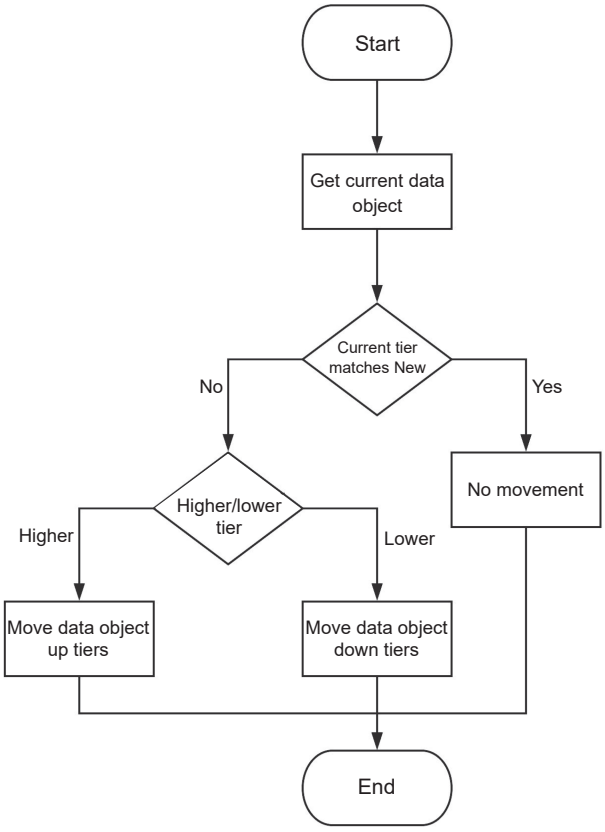


Fig. 14 Flow of the decision involved in creating a data movement action.

Algorithm 3 Moving data objects into storage

Data: sortedDataObjectList, moveAction, and moveActionList
foreach dataObject in sortedDataObjectList **do**
  if dataObject.currentTier ! = dataObject.proposedTier **then**
    moveAction = new MoveAction();
    moveAction.dataObjectId = dataObject.id;
    moveAction.targetTier = dataObject.proposedTier; Insert moveAction into MoveActionList;
  end
end
foreach each storageTier in storageTierList **do**
  migrate moveAction.dataId into StorageTier matching moveAction.targetTier;
end

We have not established any clear boundaries between movement to tiers because if working in reverse order (the lowest to highest), then the tiers will get used in the desired order anyway, even without explicitly having to establish that. This is due to the order in this section. This is demonstrated in Fig. 15, the first diagram labeled "Phase 1" shows only the lowest scoring data moving down to the lowest tier of

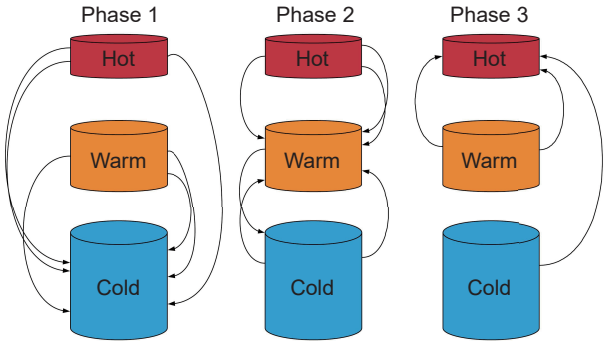


Fig. 15 Three non-explicit phases of data movement.

storage once all of this movement completes, next is the "Phase 2" diagram, slightly different here as we have the data moving down and up into the middle tier of storage. Finally, "Phase 3" is the remaining data actions moving up to the hottest tier of storage.

This example has only presented three tiers of storage for simplicity, but more tiers of storage would just result in the "Phase 2" style of movement being repeated as per the number of tiers of storage available. Once the movement action list has come to an end, so has our movement policy, and all data should now reside in its most optimal location as determined by our proposed data temperature score and method of balancing/moving data.

In Algorithm 3, there is a second loop not discussed above. This is where we action the data movement, each of the move actions generated is then integrated in reverse order, so the movement of data occurs in a downwards motion towards the colder tiers where storage is more plentiful; this is to prevent any conflicts with storage tiers being too full to accept data.

5 Experimental and Implementation Detail

The proposed system is based upon existing works not being suitable for varying scenarios; this means that our experimentation has to demonstrate that multiple use case scenarios have been considered and tested against to validate our primary objective. To achieve these tests on various use-cases, we determined that a simulated environment shown in Fig. 16 would be more suitable and allow for the testing of only the specific variables that were required without any concern for external factors influencing our results.

5.1 Experimental environment

The presented algorithms and variables will be tested on a simulator, a simulation is ideal for our current

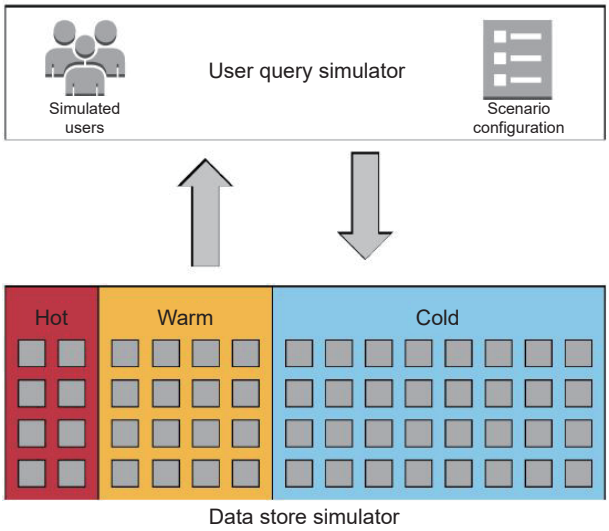


Fig. 16 Overview of the simulator.

requirements, as we can explore the potential of the proposed variables in a controlled manner based on the objectives of our research. The simulator is split into two components, one is the data and data store simulator, which represents our tiered storage infrastructure filled with data with multiple characteristics, the second is a user query simulator, which will simulate user queries against the data and data store simulator.

Individual scenarios created to test various types of usage then have an impact on how both components operate. The simulator has been programmed in JAVA. The goal of the simulator is to test the variables and algorithms proposed by our research, the success of each will be determined by two scores generated from our simulations and then evaluated. The fields are as follows:

- **Query cost:** Each simulated user query will generate a score based upon the tier of storage it has to query on, this score will be totaled. Lower score is more optimal.
- **I/O movement:** Over simulation as a whole, the amount of movement that occurs will be totaled. Lower score is more optimal.

5.2 Data store simulator

The data store simulator is our simplified representation of a storage environment with multiple tiers of storage. It initially establishes itself by inserting the number of storage tiers, the breakdown of the tiers as percentages, and the number of total data blocks across the whole data store. Data blocks are our method of representing storage, each data block has a fixed position within the data store and is assigned a temperature tier, a data block has space to assign a single data object. Data and data blocks within our simulator are all of a fixed size; this simplifies the storage space to precisely what we want to simulate and removes unnecessary complexity.

Figure 17 displays a compact version of our data store simulator. It only has three tiers of storage represented and six total data blocks across the whole data store. The following values generated using this data store simulator:

- Number of storage tiers: 3.
- Breakdown of storage tiers (percentage): 10%, 30%, and 60%.
- Total number of data blocks: 6.

To increase the size of the data store simulator, the attribute of total number of data blocks has to be increased, similarly if more tiers or differences in how the tiers are weighted then the attributes just have to be adjusted accordingly when setting up the simulator. The simulator used for our results for example uses roughly five hundred data blocks split across three storage tiers.

It was explained above that the data block's are fixed, so in Fig. 17, Data Block 1 is always within the hot tier and Data Block 5 is always located in the cold

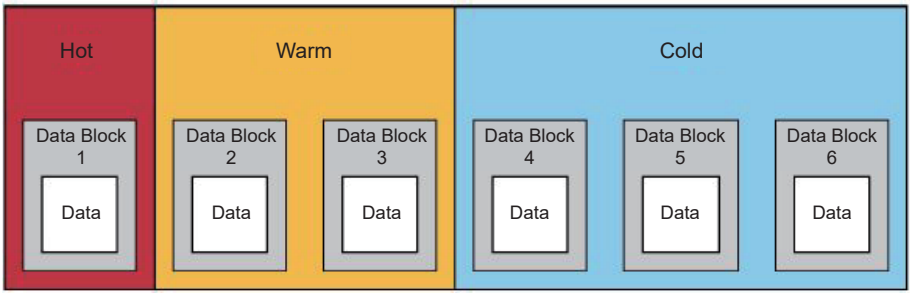


Fig. 17 Data store simulator.

tier, it is instead the data that move between data block's, each piece of data only existing within one data block. Replication is important and does massively impact query performance, but this has also been ignored to simplify storage to our current research objects.

5.3 Data object

The simulator's data object is where all features of each data item are stored but also various statistics that are used to generate various scores from the data, as explained above data objects are assigned to a data block, but can then move around between data blocks during the data temperature movement phase.

Each data object has the following features:

- **Data ID:** Unique identifier for each data object.
- **Creation DateTime:** DateTime that the data were first added into the data store simulator.
- **Data type:** Specifies the sort of data, e.g., sales and forecasting. This is used by some scenarios to target specific data.
- **Data priority:** Specific data can be essential to success so requires a higher weighting.

These are features, each data object is assigned when created. Data ID and creation DateTime are automated, but data type and data priority are governed by the scenario that is chosen. Data priority is also special in that it is also affecting the "query score" value, high priority data should be in the higher tiers of storage, to reflect the negative impact of putting high priority data in lower tiers of storage when the query cost is doubled for high priority data:

- User queries Data 16 located in warm tier.
- User queries high priority Data 2022 located in warm tier.
- User queries Data 18 located in hot tier.
- User queries high priority Data 2032 located in hot tier.

Each data object also captures the following stats of the data:

- **Access count:** Count the number of times the data object has been queried against.
- **Last accessed DateTime:** DateTime of the last query on the data object.
- **Movement counter:** Track the number of times the data object has been moved between tiers of storage.
- **Last movement:** Track whether the last movement was up a tier or down a tier.

These stats about the data are gathered or generated throughout the query process and are then used to determine a data object's individual data temperature value.

Further details of how each feature/statistic of the data is used to determine temperature will be covered in detail within the data temperature section, as its usage can change between implementations.

5.4 User query simulator

The user query simulator is the component of our simulator that manages all the users and the queries they make. Similar to the data store simulator, the actions of the user query simulator are constrained by what is established within the current scenario. The scenario tells the user query simulator how many users total will be created, the min/max range for the number of queries they will make, specific target data criteria, and the ability to specify specialist users that have their own associated rules.

5.4.1 User

The simulator has consideration for two types of users, the first a standard user and the second a high priority user. Practically they do the same thing, but depending on the specific scenario in place, high priority users will have additional rules. High priority users represent system users that are working on particular projects or where their work is mission critical to a business's operations. The defining difference is that priority users double the "query cost" value, so the lower tier of storage they have to query to, then the query cost will be much higher.

5.4.2 Query

Queries within the simulator are simple, User No. 1 targets data object A and that constitutes a query for our purposes, no actual data exists, so a simple interaction between the user and the data object are all that is needed (the primary action of the interaction is that of updating the statistics within the data object).

The data object that a user query is pretty much random, and so could be any of the data objects that have been added into the data storage simulator. The only time this is not the case is when the scenario dictates that only certain types of data should be queried by a specific type of user. Then the user is instead restricted to choosing data objects of that type or potentially another feature that is not data type.

5.5 Data temperature variable configuration

Our proposed solution is that of making use of suitable variables and conditions relevant to specific data usage scenarios, instead of the one size fits all approach. Thus, we prepare a selection of variables to be used in our simulated environment. Each proposed configuration of variables and conditional rules has been designed to the best match the scenario outlined below. The variables and conditional rules for each variable configuration (VC) alongside an explanation on the motivations for combining such variables and what the proposed outcome are in the rest of this section.

5.5.1 Variable Configuration 1 (VC1)

- Variables:
  - Age [ranged].
  - Usage frequency [ranged].
- Conditional rules:
  - None.

This configuration of variables is to act as a control and represents the prevalent data temperature variables currently used as standard. Usage frequency is the combination of read and write activity upon the data objects.

5.5.2 Variable Configuration 2 (VC2)

- Variables:
  - Who accessed [binary].
  - Frequency of usage [ranged].
- Conditional rules:
  - None.

This configuration is focused primarily on usage and who the data are being used by to determine what is important.

5.5.3 Variable Configuration 3 (VC3)

- Variables:
  - Write frequency [ranged].
- Conditional rules:
  - If day is weekday or weekend [binary].

In this configuration, writes are a priority, and we know data are used different between weekends and weekdays.

5.5.4 Variable Configuration 4 (VC4)

- Variables:
  - Age [ranged].
  - Who accessed [binary].
- Conditional rules:
  - Data type equals “data type A” [binary].

Variable Configuration 4 is focused towards a specific data type, but has been combined with age and who accessed, so the data age out over time and with considerations for if high priority users are requiring the data.

5.5.5 Variable Configuration 5 (VC5)

- Variables:
  - Read frequency [ranged].
- Conditional rules:
  - Tag equals “Tag A” [binary].
  - Tag equals “Tag B” [binary].
  - Tag equals “Tag C” [binary].

Reading of data objects is more important, with priority being given to three specifically tagged data objects.

5.5.6 Variable Configuration 6 (VC6)

- Variables:
  - Age [ranged].
  - Usage frequency [ranged].
  - Created by [binary].
  - Size [ranged].
  - Who accessed [binary].
- Conditional rules
  - Data type equals “data type A” [binary].
  - Tag equals “Tag C” [binary].
  - If day is weekday [binary].

Where Variable Configuration 1 was the control, Configuration 6 is an extreme and has just had multiple variables bundled together with no specific goal in mind. Unlike the other configurations, it does not have a matching scenario.

5.5.7 Variable Configuration 7 (VC7)

- Variables:
  - Any.
- Conditional rules:
  - Any.

The six prior variable configurations above were planned and served a purpose. This configuration instead will pseudo-randomly select any/all variables and conditional rules featured in the six configurations above.

5.6 Use case scenarios

This research targets varying types of user and data access patterns, meaning we cannot adequately test against a single kind of scenario where all data and users are equal, we instead propose various scenarios to test our variables and algorithms. The individual scenarios have been constructed with real-world use cases in mind but also with considerations for the

variables we have proposed; the scenarios are relevant because they for one indicate what sort of scenario each variable/algorithm excels at but also where they fail to perform and potentially hinder performance.

The scenarios described below first present the configuration and established rules of the scenario before explaining the use case it represents and the specific goals the scenario aims to test.

5.6.1 Scenario 1

This scenario represents the current standard and will act as our control. The latest data will be queried more often and some data items will be frequently queried, even if not of the latest dataset.

Scenario features:

- 100% standard users.
- New data are added into the scenario after every simulated day.
- User queries will have a preference towards new data objects.
- A set of data objects (First 10% of the original dataset) will repeatedly be queried regardless of newness.
- Recommended configuration: Variable Configuration 1.

5.6.2 Scenario 2

Scenario 2 is demonstrative of an environment where there is a smaller group of users that require high priority usage over the rest of users. Data are not aged out in this scenario, instead it focuses on the usage of data and who it is being used by.

Scenario features:

- 80% standard users and 20% high priority users.
- New data are not added in this scenario.
- Recommended configuration: Variable Configuration 2.

5.6.3 Scenario 3

Scenario 3 is representative of a high write environment, the data being written take a higher priority than data that are being read. The scenario also has much different usage of data depending on whether it are a weekday or not, data written at the weekend are much more important than that the data written within the weekday.

Scenario features:

- New data are not added in this scenario.
- Scenario has a higher number of writes than reads (60/40 split).
- Recommended configuration: Variable Configuration 3.

5.6.4 Scenario 4

Scenario 4 is interested in newer data, that is influenced by who accessed it and if it is off a specific data type.

Scenario features:

- New data are added into the scenario after every simulated day.
- Recommended configuration: Variable Configuration 4.

5.6.5 Scenario 5

This scenario is heavily influenced by users tagging specific data objects, certain tags are then flagged as important. In this scenario, the tagged objects will have a higher priority to our users.

Scenario features:

- New data are not added in this scenario.
- Data objects can have multiple tags.
- Recommended configuration: Variable Configuration 5.

5.7 Simulating data temperature

Simulation combines the variable configuration, scenario, storage, and user query simulator being used together. The variable configuration and the scenario establish the rules based on which these simulators will act. For example, Scenario 2 applied to the user query simulator tells it to generate 80% standard users and 20% flagged as high priority, other than being flagged differently, they will act the same as other users unless a rule specifies otherwise. The specialist users will not impact data temperature values unless we apply a data temperature variable configuration that is specifically targeting that type of user.

5.8 Scoring

Scoring is applied in two ways, one is based upon queries and the second is based upon data movement. A total score is given that combines them, but the individual scoring are evaluated in Section 6 also. In our testing, a low score is always better. Query-based scoring is pretty much that a simulated user query is performed, and a score is awarded. The value of the score is based upon the tier of storage that is being queried, if the data object being queried is more accessible to the user, i.e., in the hotter tiers of storage then the better, so a lower score is awarded, e.g., user queries a data object in the cold tier of storage score for that query would be a 3, next user queries a data object

in the hot tier and scores a 1 (assuming a 3-tier hot, warm, and cold architecture).

Movement-based scoring instead looks at how much data are actually moved around during the balancing process, data should be moving around, but excessive movement is detrimental to system performance, so should be reduced where possible.

6 Result and Discussion

Using the simulators outlined in Section 5.1, we have gathered the results presented here. We constructed five scenarios, with each scenario designed to simulate a distinct type of system usage and user activity. Each scenario has its own optimal variable configuration, but we will test each variable configuration against each scenario, in addition to random (using pseudo-randomly selected variables) and extreme variable configurations (using multiple variables with no targeted benefit).

The first scenario and variable configuration is under our control, as it is based on the current data temperature, the age and frequency of data usage, and the usage of data. Random is not a specific configuration like the others; rather, score values were assigned pseudo-randomly to the data positions and used in lieu of the full process carried out for the other configurations.

We present the results from the perspective of a scenario, with the most desirable outcome being that the variable configuration designed for it is the optimal one. The worst outcome should, in theory, be the random or extreme configuration, but it could easily be one of the other configurations, as each could be configured in a way that benefits its target scenario but not the others.

In this section, the authors describe the outcomes of their study's variable configurations. The first scenario serves as the baseline and is based on the existing data temperature method, which takes into account data age and usage frequency. A new configuration referred to as "random" is introduced in which score values are assigned to data positions pseudo-randomly, as opposed to the full process used by the other configurations. The results are presented from the vantage point of a particular scenario, with the optimal variable configuration tailored to it. The total score, which is comprised of scores for data movement and data position at query time, the better the outcome. Stackable column charts are used to illustrate the query and movement scores for each variable configuration.

Continuous querying of data with a high temperature score would result in its placement in a "hot" storage location, where it would remain unmoved during score sorting and balancing. The addition of new data and user preference for new data keeps costs low, as new data initially receive the highest score and are swiftly placed in a hot or warm tier, reducing the cost of querying new data. Even though maintaining older data and prioritizing new data are contradictory variables, the query cost remains low. If new data were initially stored on the lowest tier, query and movement expenses would increase. It is more cost-effective to gradually age data downward as it is balanced within the system as opposed to rapidly moving it up and down as new data accumulate behind it.

6.1 Scenario 1

This scenario represents the current standard and acts as our control. The latest data were queried more often, and some data items were frequently queried, even if they were not in the latest dataset. The optimal configuration for this scenario was Variable Configuration 1.

In Fig. 18, for Scenario 1, we see that the optimal configuration of VC1 scores the best overall, followed closely by VC2 and VC4. The VC1 scoring the lowest is expected because this is the scenario it should perform best in, as it is designed to accommodate the variables of new data and a specific set of data that is frequently requested. It scored very low on the query cost and then relatively low on movement, topped only by VC4's lower score of 62. The low query score of 28 in VC1 is likely due to the specific set of data continually being queried, despite new data being added, this maintains a set of data that are constantly low cost to query and thus provides a lower result overall.

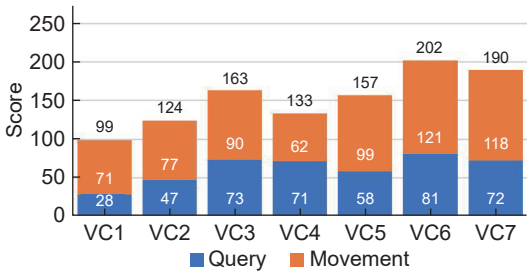


Fig. 18 Scenario 1 query and movement stacked column graph.

The continual querying would maintain an overall high temperature score and result in this continually queried data being allocated a "hot" storage location, a location that the data would most likely already be sorted in, so it would require zero movement when the overall scores were sorted and balanced. It would still maintain its "hot" position or at least movement from a "warm" tier position. The addition of new data and the user prevalence for new data would also keep this cost low. Data would get added and score the highest due to being the newest, so the sorting/balancing of the proposed process would place it into a hot/warm tier immediately, reducing the cost of any new data queried.

The query cost of 71 is still low considering the two variables are opposed, one maintaining the older data, the other forcing new data to be given priority. The majority of the movement is, no doubt, due to the new data coming into the system and getting pushed down. If the latest data were initially stored in the lowest tier, they would lead to increased query and movement costs. This is because the initial query would be high, accessing lower-scoring cold tiers. Additionally, the cost of moving the data up, only to have it quickly move back down with new rising data, would result in more overall movement compared to a system where data age gradually and are balanced within the tiers.

VC2 and VC4 were also making use of traits used in VC1 such as age or frequency of usage, so they shared a similar boost to performance. The low query score of VC2 is for the same reason as VC1, and its benefit of data continuously scoring high due to its repeated usage. Meaning less movement spent on moving data around and having to have it rebalanced, also equaling more time with data in an optimal location and reducing query cost. VC4 received a similar benefit but has a much higher movement score due to the fact that instead of data being maintained in position even with new data being added, it instead has to continually move new data down tiers as it gets added into storage to maintain the optimal data position and a low query score.

The configurations of VC3 and VC5 did not perform much worse, which is slightly surprising given the score distance between them and configurations VC6 and VC7, which scored considerably worse. Both VC3 and VC5 use a sub-typing of usage, which are read and write. For this scenario, part of the time they would get some benefit out of usage, but the other half they would not get any benefit due to the wrong type of data usage encountered. This would explain the higher movement cost, as data would see some read/write usage and be sorted into one location, but then the usage would be very different shortly after and see the data moving out of its assigned tier. Thus increasing the cost of the query overall.

The mismatch of so many variables for VC6/VC7 with only two simple inputs meant that the scoring process was handing out both high and low temperature scores to all new incoming data, meaning the new data were not sorted immediately into the higher tiers and received a much higher cost to query overall.

One of our primary points is that we should not only be focused on usage/time variables for calculating data priority, but in the context of this scenario, the only valid variables are usage/time, so they are what is used, and they are what performs the best overall. So it is important to consider that usage and time are not the only variables, but in specific instances they can be the only valid choice to make.

6.2 Scenario 2

Scenario 2, as depicted in Fig. 19, is demonstrative of an environment where there is a smaller group of users that require high priority usage over the rest of users. Data are not aged out in this scenario, instead, it focuses on the usage of data and who it is being used by. The optimal configuration for this scenario was Variable Configuration 2.

The scenario is very focused on satisfying a very specific group of users, this would probably explain why the score is as low as it is for VC2 as it is a very focused and niche set of scoring requirements and conditions. Coming in the second was VC4, this is again due to this configuration also including the same prevalence towards the higher priority users that are

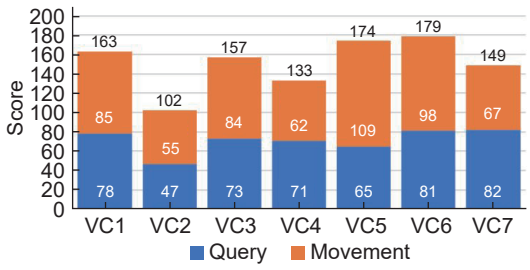


Fig. 19 Scenario 2 query and movement stacked column graph.

accessing the system, the other fields used in VC4 would have decreased the score in this instance. The remaining configurations are pretty consistent, hitting around 130–140, this is strange for VC6 and VC7, we can change the 121 scores from VC7 up to random chance, but it is strange that the chaos of VC6 did as well as it did for this much more user-focused scenario.

6.3 Scenario 3

Scenario 3, as depicted in Fig. 20, is representative of a high write environment, the data being written take a higher priority than data that are being read. The scenario also has much different usage of data depending on whether it is a weekday or not, data written at the weekend are much more important than the data written within the weekday. The optimal configuration for this scenario was Variable Configuration 3.

VC3 performed the best overall, with much lower query and movement scores than the closest configuration by 23 points, with VC6 scoring close to double that of VC3. VC3 is the most appropriate configuration, but we would not have expected it to perform this well, the scenario does have a slight preference for data writes, but this would not reflect such a drastic scoring variance.

VC7 also performed strangely well, this is most likely due to a random selection of a write preferential "conditional rule", otherwise we would expect it to have performed very similarly to VC6.

The other VCs performed similarly within a range of 20 points of each other, VC2 performed considerably worse than the others, but nothing too concerning that would warrant further investigation.

6.4 Scenario 4

Scenario 4, as depicted in Fig. 21, is interested in newer data, that is influenced by who accessed it and if

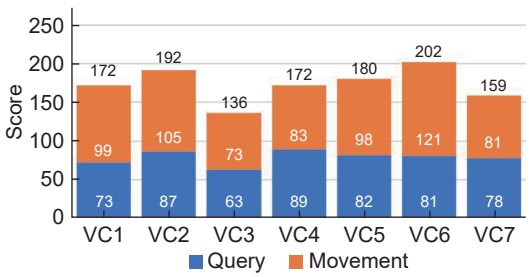


Fig. 20 Scenario 3 query and movement stacked column graph.

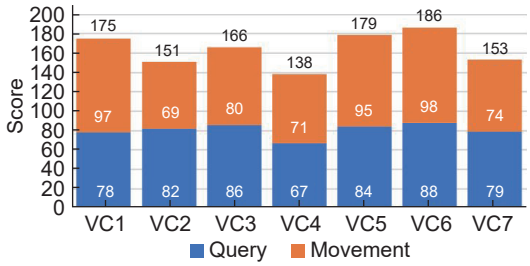


Fig. 21 Scenario 4 query and movement stacked column graph.

it is off a specific data type. The optimal configuration for this scenario was Variable Configuration 4.

VC4 performed the best, scoring lowest in both query and movement, this will be due to the usage of both "age" and "who accessed" variables giving data priority. This will also be the reason VC2 performed well also, as it also uses the "who accessed" variable to determine priority.

The outliers for this scenario would be VC1 and VC7, VC1 because it has the "age" variable, and so would have expected it to perform similarly to VC2 and not score as high as it did. VC7 because it managed to score so low despite being a random selection of variables similar to VC6, which performed the worst overall.

6.5 Scenario 5

Scenario 5, as depicted in Fig. 22, is heavily influenced by users tagging specific data objects, certain tags are then flagged as important. In this scenario, the tagged objects will have a higher priority for our users. The optimal configuration for this scenario was Variable Configuration 5.

An immediate and noticeable difference across these results compared to the others is that they are all slightly lower values than the previous results, this, we presume, is in part due to the lack of "new data" being added.

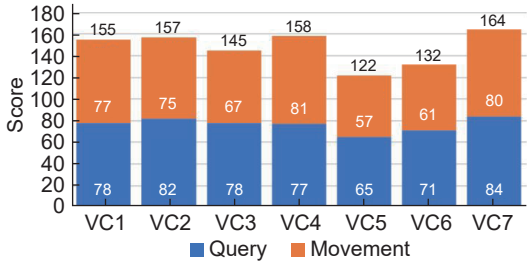


Fig. 22 Scenario 5 query and movement stacked column graph.

VC5 performed the best with the lowest results overall in both query and movement scores, this was expected with the heavily tag-focused scenario, resulting in a lower query cost and less movement overall. This was strangely followed very closely by VC6 our "extreme" implementation of a varied assortment of attributes. VC6 does use tags, so by chance, the scenario potentially favored this specific type of "Tag C" that VC6 used.

The other variable configurations performed around the 150–160 range with very similar query and movement scores. Surprisingly, VC3 performed much better than the others with no discernible reason, as no tagging or benefit to new objects is offered in VC3. Another surprise is that VC7 performed worse, even though it was using a random selection of attributes and conditions. We would have thought it would be in line with the scoring of VC1, VC2, and VC4 at least.

7 Conclusion and Future Work

The primary objective of this work was to present a solution that combined numerous user and system variables, allowing for more performance-relevant data placement within tiered-storage solutions. Essentially, we were presenting an extended definition of "data temperature" that encapsulated more than just age and usage variables.

The presented variables and algorithms have provided a clear benefit when used with the scenario that they have been specifically configured for, compared to the control, or when used against an alternative scenario. This was an expected outcome, and a different result would have been concerning for the proposed research. The existing simple variables of age and usage that acted as the control performed well and even beat some proposed variable configurations, not considerably, but this was either due to too much movement or not enough due to how the weighting of what should and should not move worked out.

Our research promotes a more bespoke attitude toward data movement. Similar to choosing a specific database type or storage mediums that match specific requirements, the choice of how your data move throughout a storage solution should also be presented as a point of customization in an effort to get the most benefit out of your tiered storage solution.

The primary issue with the research as presented at present is that it has been tested on a simulation and not a real working system being used by actual users. The experiment presented has attempted to propose a simplified abstraction of a real system, but a simulation can never really hold up to experiments in reality. The simulator did provide it with a relatively controlled environment to test out various configurations in a convenient and simpler manner, making it ideal at this current level of research.

A potential limitation of the research to system performance is that by complicating the process with too many or conflicting variables, this in turn would hinder system performance by either data not being moved where it is most optimal, or even too much movement occurring, which in turn would hinder performance. Admittedly, for a majority of use cases, a standard data temperature of age and usage, will work admirably, but for more specific use cases where priority data usage can not be determined by age or usage alone, we see more benefit from our proposed variables and algorithms.

Other topics worthy of exploration include identifying additional variables not covered in this work that could be used to gauge a dataset's temperature. Further validation and testing are required, with real world case studies being the most ideal. This also prompts research into the discovery of metadata characteristics unique to an individual case study, as not all variables will be appropriate for all applications. Our proposed solution presented an opportunity for greater user control and input in determining a data's priority and ideal location, but as was discussed above, greater user control can also result in a decrease in performance, not even considering the overhead and cost of managing and implementing these additional data temperature variables, so this is something worth exploring further.

In future, the investigations are also required into machine learning to determine future and expected patterns in priority, working towards a more proactive system, and ensuring data is where it needs to be prior to users actually needing it there. We also need to investigate the effect that the number of tiers has on data temperature, the different weights that need to be applied to reflect these changes, the frequency of data movement, and the impact on I/O costs when data move too much between tiers.

References

[1] J. M. Tien, Big data: Unleashing information, *J. Syst. Sci.*

Syst. Eng., vol. 22, no. 2, pp. 127–151, 2013.

[2] T. R. Gregory, Synergy between sequence and size in large-scale genomics, *Nat. Rev. Genet.*, vol. 6, no. 9, pp. 699–708, 2005.

[3] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling, in *Proc. 5th European Conf. Computer Systems - EuroSys '10*, Paris, France, 2010, pp. 1–14.

[4] V. Roussev, G. Richard, and D. Tingstrom, dRamDisk: Efficient RAM sharing on a commodity cluster, in *Proc. IEEE Int. Performance Computing and Communications Conf.*, Phoenix, AZ, USA, 2006, pp. 193–198.

[5] J. Guerra, H. Pucha, J. Glider, W. Belluomini, and R. Rangaswami, Cost effective storage using extent based dynamic tiering, in *Proc. 9th USENIX Conf. File and Stroage Technologies* (*FAST'11*), San Jose, CA, USA, 2011, pp. 1–14.

[6] D. Basin, E. Bortnikov, A. Braginsky, G. Golan-Gueta, E. Hillel, I. Keidar, and M. Sulamy, KiWi: A key-value map for scalable real-time analytics, *ACM Trans. Parallel Comput.*, vol. 7, no. 3, p. 16, 2020.

[7] V. Sundaram, T. Wood, and P. Shenoy, Efficient data migration in self-managing storage systems, in *Proc. IEEE Int. Conf. Autonomic Computing*, Dublin, Ireland, 2006, pp. 297–300.

[8] G. Zhang, L. Chiu, and L. Liu, Adaptive data migration in multi-tiered storage based cloud environment, in *Proc. IEEE 3rd Int. Conf. Cloud Computing*, Miami, FL, USA, 2010, pp. 148–155.

[9] J. Zhang, M. Ma, W. He, and P. Wang, On-demand deployment for IoT applications, *J. Syst. Archit.*, vol. 111, p. 101794, 2020.

[10] P. Gupta and M. Pegah, A new thought paradigm: Delivering cost effective and ubiquitously accessible storage with enterprise backup system via a multi-tiered storage framework, in *Proc. 35th Annual ACM SIGUCCS Fall Conf.*, Orlando, FL, USA, 2007, pp. 146–152.

[11] R. Buyya, S. K. Garg, and R. N. Calheiros, SLA-oriented resource provisioning for cloud computing: Challenges, architecture, and solutions, in *Proc. Int. Conf. Cloud and Service Computing*, Hong Kong, China, 2011, pp. 1–10.

[12] J. Tai, B. Sheng, Y. Yao, and N. Mi, Live data migration for reducing SLA violations in multi-tiered storage systems, in *Proc. IEEE Int. Conf. Cloud Engineering*, Boston, MA, USA, 2014, pp. 361–366.

[13] R. Rizzi and D. Cariolaro, Polynomial time complexity of edge colouring graphs with bounded colour classes, *Algorithmica*, vol. 69, no. 3, pp. 494–500, 2014.

[14] I. Robertson-Steel, Evolution of triage systems, *Emerg. Med. J.*, vol. 23, no. 2, pp. 154–155, 2006.

[15] M. Christ, F. Grossmann, D. Winter, R. Bingisser, and E. Platz, Modern triage in the emergency department, https://www.aerzteblatt.de/int/archive/article/79788, 2010.

[16] M. Yamada and S. Yamaguchi, Filesystem layout reorganization in virtualized environment, in *Proc. 9th Int. Conf. Ubiquitous Intelligence and Computing and 9th Int. Conf. Autonomic and Trusted Computing*, Fukuoka, Japan, 2012, pp. 501–508.

[17] Z. Yang, Y. Wang, J. Bhamini, C. C. Tan, and N. Mi, EAD: Elasticity aware deduplication manager for datacenters with multi-tier storage systems, *Clust. Comput.*, vol. 21, no. 3, pp. 1561–1579, 2018.

[18] G. Zhang, L. Chiu, C. Dickey, L. Liu, P. Muench, and S. Seshadri, Automated lookahead data migration in SSD-enabled multi-tiered storage systems, in *Proc. IEEE 26th Symp. on Mass Storage Systems and Technologies* (*MSST*), Incline Village, NV, USA, 2010, pp. 1–6.

[19] W. Shin, C. D. Brumgard, B. Xie, S. S. Vazhkudai, D. Ghoshal, S. Oral, and L. Ramakrishnan, Data jockey: Automatic data management for HPC multi-tiered storage systems, in *Proc. IEEE Int. Parallel and Distributed Processing Symp.* (*IPDPS*), Rio de Janeiro, Brazil, 2019, pp. 511–522.

[20] S. Sankar and K. Vaid, Storage characterization for unstructured data in online services applications, in *Proc. IEEE Int. Symp. on Workload Characterization* (*IISWC*), Austin, TX, USA, 2009, pp. 148–157.

[21] S. Rawson, M. G. Iadanza, N. A. Ranson, and S. P. Muench, Methods to account for movement and flexibility in cryo-EM data processing, *Methods*, vol. 100, pp. 35–41, 2016.

[22] A. Kala Karun and K. Chitharanjan, A review on Hadoop—HDFS infrastructure extensions, in *Proc. IEEE Conf. Information & Communication Technologies*, Thuckalay, India, 2013, pp. 132–137.

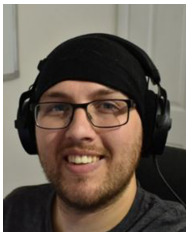
[23] P. K. Acharya and S. K. Patro, Effect of lime and ferrochrome ash (FA) as partial replacement of cement on strength, ultrasonic pulse velocity and permeability of concrete, *Constr. Build. Mater.*, vol. 94, pp. 448–457, 2015.

Ashiq Anjum is a professor of distributed systems at University of Leicester, UK, and the director of enterprise and impact for the School of Computing and Mathematical Sciences, University of Leicester, UK. His areas of research include data intensive distributed systems, distributed machine learning models, and physics informed machine learning models for digital twins. He has been investigating digital twins to emulate the real time behaviour of (bio)mechanical and cyber-physical systems.

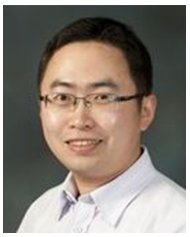
Nick Antonopoulos is a professor in cloud computing at Edinburgh Napier University, UK. He has more than 20 years of academic and leadership experience, with a very strong background in initiating, leading and delivering improvements at an institutional level. He has an excellent international reputation in his field, evidenced by his papers, books, chairing of prestigious conferences, and his active leadership of broad reaching research partnerships nationally and internationally.

Ali Zahir is an accomplished researcher and industry professional. He currently is pursuing the PhD degree at University of Leicester, UK, with a specialization in distributed computing. His research focuses on elevating data processing workflows in cloud and edge computing environments, aiming to optimize the handling of data-intensive scientific analyses. He is deeply engaged in various national and international research and development projects in the fields of high performance computing (HPC), data science, networking, system design and architecture, etc.

Muhammad Usman Yaseen received the master degree from Dalarna University, Sweden, in 2012, and the PhD degree from University of Derby, UK, in 2019. He is an assistant professor at COMSATS University Islamabad, Pakistan. With a rich background in both higher education and industry, he has accumulated over 15 years of expertise in machine learning, deep learning, and computer vision. He has a strong track record of developing innovative deep learning models for analyzing real-world data sources like images, videos, and text.



Dominic Davies-Tagg is a software developer at Steris. He also worked as a researcher at University of Derby, UK. His research interests include big data computing, databases, and distributed computing.



Lu Liu is a professor at the Department of Informatics, University of Leicester, UK, with expertise in artificial intelligence, data science, and the Internet of Things, focusing on developing trustworthy and sustainable systems based on machine learning for health, net zero, and digital manufacturing. He has been investigating machine learning models for large-scale sustainable information and communications technology (ICT) systems for many years and successfully developed several AI-driven methods for reducing the carbon footprint of data centres and communication systems.