# GAMPS: Compressing Multi Sensor Data by Grouping and Amplitude Scaling

Sorabh Gandhi[*]
UC Santa Barbara
sorabh@cs.ucsb.edu

Suman Nath
Microsoft Research
sumann@microsoft.com

Subhash Suri[†]
UC Santa Barbara
suri@cs.ucsb.edu

Jie Liu
Microsoft Research
liuj@microsoft.com

## ABSTRACT

We consider the problem of collectively approximating a set of sensor signals using the least amount of space so that any individual signal can be efficiently reconstructed within a given maximum ($L_\infty$) error $\varepsilon$. The problem arises naturally in applications that need to collect large amounts of data from multiple concurrent sources, such as sensors, servers and network routers, and archive them over a long period of time for offline data mining. We present GAMPS, a general framework that addresses this problem by combining several novel techniques. First, it dynamically groups multiple signals together so that signals within each group are correlated and can be maximally compressed jointly. Second, it appropriately scales the amplitudes of different signals within a group and compresses them within the maximum allowed reconstruction error bound. Our schemes are polynomial time $O(\alpha, \beta)$ approximation schemes, meaning that the maximum ($L_\infty$) error is at most $\alpha\varepsilon$ and it uses at most $\beta$ times the optimal memory. Finally, GAMPS maintains an index so that various queries can be issued directly on compressed data. Our experiments on several real-world sensor datasets show that GAMPS significantly reduces space without compromising the quality of search and query.

## Categories and Subject Descriptors

E.2 [**Data Storage Representation**]: Composite structures; H.3.2 [**Information Storage and Retrieval**]: Information Storage

## General Terms

Algorithms, Performance, Theory

[*]Part of this work was done when the author was at Microsoft Research. This research was supported in part by the National Science Foundation Grants CCF-0702798 and CNS-0626954.

[†]This research was supported in part by the National Science Foundation Grants CCF-0702798 and CNS-0626954.

## Keywords

multi-sensor data compression, clustering

## 1. INTRODUCTION

Recent advances in sensing technologies have made possible, both technologically and economically, the deployment of densely distributed sensor networks. For instance, the Microsoft DCGenome project [23] deploys a large number of sensors in a data center to continuously monitor the physical environment by measuring temperature and humidity around servers. To provide near real-time visibility of the physical conditions at the data center, the sensing must be frequent, perhaps every few seconds, so that one can respond quickly to abnormal conditions such as sudden temperature spikes. It is easy to see how this could lead to a data glut: even a few thousand sensors, recording a few tens of bytes per second would generate tens of gigabytes *each day.* Archiving these data over a few years for historical comparison and trend analysis would consume terabytes to petabytes of storage and bandwidth. To minimize the overhead of storing, managing and sharing these sensor data, therefore, we must apply smart approximation schemes that significantly reduce their size without compromising our monitoring and analysis abilities. Such a solution can benefit many other applications such as a server farm monitoring system that collects and archives various system counters (e.g., CPU usage) from a large number of servers, an Internet traffic engineering system that collects/archives various traffic flow related counters from a large number of routers, and so on. For many useful data mining tasks, such as analyzing and forecasting resource utilization, anomaly detection, and forensic analysis, compressed data must guarantee a given maximum ($L_\infty$) decompression error.

An individual sensor's measurements can be thought of as a time series (or, a signal), and there are many techniques known for *compressing* a single time series [1, 4, 5, 27, 10, 15, 17, 14, 21]. Our interest, however, is complementary to this existing body of work—we focus on collectively compressing data from multiple sensors for space-efficient archiving and time-efficient querying. In other words, rather than compressing a single time series of measurements, we wish to consider the problem of compressing multiple time series simultaneously, taking advantage of the correlation among them. Unlike the single series case, however, not only is very little known for compressing multiple signal streams, but even the problem is somewhat difficult to formalize cleanly. In this paper, we make a principled attempt to formalize the multi-sensor compression problem and propose schemes with worst-case error guarantees.

To be precise about our goals and techniques, let us emphasize that we are interested in *lossy* but *combinatorial* methods with the following general desiderata: (1) to approximate sensors' data with a worst-case guarantee on the approximation errors, (2) to deal with $L_\infty$ norm (maximum) error, as opposed to cumulative type errors, such as $L_2$, or $L_1$, because $L_\infty$ approximation better preserves local spikes, which are extremely important in applications intended for local anomaly detection [23], (3) to support query-processing directly on compressed data, so methods that use compression primarily for communication and necessitate *decompression* before queries are not of interest. While we focus on efficient archiving of sensor data at a central location, we do not directly address the important issue of transferring data from sensors to the central location in this paper. One can use existing energy-efficient data collection techniques (e.g., [9]) if the sensors are energy-constrained.

There are several schemes that achieve the three goals stated above for an individual signal, most notably [3]. However, achieving them all for multiple signals is challenging for several reasons. First, one must decide which signals to compress together; trying to compress uncorrelated or weakly correlated signals together may yield suboptimal result. In a small network, such *grouping* can be manually decided based on signal correlations. However, with a large number of sensors, this needs to be done automatically—the compression system must automatically detect correlated signals such that grouping them together yields maximal compression. This is further complicated by the fact that the correlation among signals can change over time due to changes in the physical environment [23], and thus the compression system must be able to *dynamically* determine efficient grouping.

Second, interestingly, while many signals from physical sensors differ significantly in values, they often show a remarkable similarity in *form*. More precisely, even if two signals significantly differ in their absolute values, it might be possible to apply a *transformation* on them such that the transformed signals, along with a description of the transformation, can be efficiently compressed together. We explain this property and the physical principle behind this in Section 4. Finding such transforms in full generality is out of the scope of this paper, but we show that a simple linear transform, which we call *amplitude scaling*, works well for many real-world datasets.

Our main result is GAMPS (Grouping and AMPlitude Scaling), a general framework to address the above two challenges. First, it dynamically groups signals such that compressing signals within each group together yields a maximal compression ratio. To find the optimal grouping of signals, we map the problem to the facility location problem [2, 18]. Second, it uses amplitude scaling transformation to efficiently compress even the signals which differ significantly in values. Additionally, it guarantees $L_\infty$ norm (maximum) error, as opposed to cumulative type errors, such as $L_2$ or $L_1$. Finally, GAMPS maintains an index of compressed signals so that several useful queries can be efficiently answered without uncompressing entire signals.

Our theoretical results include a combinatorial formulation of the multi-sensor data compression problem and polynomial time schemes with a worst-case quality of approximation. We use a standard definition of $(\alpha, \beta)$ approximation in combinatorial optimization. Namely, suppose that the amount of memory used by the *optimal* algorithm to guarantee an $L_\infty$ error of $\varepsilon$ is $OPT$. Then, an $(\alpha, \beta)$ approximation algorithm takes memory at most $\beta$ OPT and guarantees an error at most $\alpha\epsilon$. In particular, for compressing the sensor data *directly* (without scaling and grouping), we show a polynomial time algorithm that achieves $(5, O(\log k + \log(OPT))$ approximation for representing $k$ signals using piecewise constant templates. With amplitude scaling and grouping, we show a polynomial time algorithm that achieves $(3 + \triangle, O(\log n))$ approximation for representing $k$ signals with piecewise constant ratio and template signals, where $\triangle$ is the worst case ratio of signal values at any time $t$, where $1 \leq t \leq n$.

In summary, we make the following contributions:

1. We formalize the multi-sensor compression problem as a combinatorial optimization problem and provide a polynomial-time scheme with a worst-case quality of approximation.

2. We propose GAMPS, a novel framework to compress a large number of sensor data streams. Unlike most existing work, GAMPS guarantees a given worst-case maximum ($L_\infty$) error. Notable features of GAMPS include (i) dynamic discovery of groups of sensor signals that can be maximally compressed together, (ii) use of appropriate amplitude scaling to further improve overall compression ratio of signals, and (iii) an index of compressed data that enables answering several important queries directly from compressed data. To the best of our knowledge, GAMPS is the first system to provide these attractive features.

3. We evaluate GAMPS with several real-world datasets. Our evaluation shows significant space-efficiency of GAMPS over the state-of-the art $L_\infty$ error approximation schemes for individual signals.

The rest of the paper is organized as follows. Section 2 discusses related work. We present our solution for multi-sensor compression in Section 3, for a simpler scenario where sensors share values, and in Section 4, for a more general scenario. Section 5 describes how GAMPS support queries on compressed data. Finally, we evaluate GAMPS in Section 6.

## 2. RELATED WORK

Time-series approximation is a hot topic in research. Researchers have proposed using a variety of techniques including Discrete Fourier Transformation (DFT) [27, 1], Discrete Wavelet Transformation [4], Singular Value Decomposition (SVD) [10], Discrete Cosine Transformation (DCT) [17], Piecewise Aggregate Approximation (PAA) [15], Adaptable Piecewise Constant Approximation (APCA) [14], Indexable Piecewise Linear Approximation (PLA) [5], Symbolic Aggregate Approximation (SAX) [21], etc. However, all these techniques are designed for approximating a single signal, and the focus of our work is joint approximations across multiple signals. Among these, DCT, DFT and SVD are $L_2$ based, while our focus is on bounded $L_\infty$ error. Wavelets have an $L_\infty$ variant [11], but it is not clear how this can be extended to share coefficients across signals. We focus on using piecewise approximations, and in particular piecewise constant approximations, but all the techniques presented here apply to PAA, PLA, and SAX as well.

Several recent work have proposed solution for compressing sensor data while exploiting correlation (see [16] for a survey). Unlike our work, most of them consider $L_2$ metric, focus on reducing communication overhead, and consider compressing only a single sensor signal exploiting temporal correlation. SBR [9] and RIDA [7], like ours, compress multiple sensor signals. However, unlike GAMPS, they are based on $L_2$ metric (SBR can support $L_\infty$ only with a probabilistic guarantee and expensive computation), require similar signals to be statically grouped together before running the algorithms, and do not support query processing on compressed data.

Like this work, distributed source coding (DSC) aims to compress signals from multiple sources together [29, 31, 33]. However, our requirement is less stringent than classical DSC, as our compression is centralized. Unlike GAMPS, DSC schemes are either lossy or primarily $L_2$ based, and processing queries on data compressed with source coding is very expensive.

There is a vast literature on grouping (or clustering) time series (e.g., see [22, 25] for related work). The primary goal of grouping in GAMPS is different from that of existing techniques: it needs to group those signals together that can share representations for maximal compression. Therefore, our grouping algorithm is different from the existing techniques.

Many existing time-series index structures enable fast detection of similar series under a variety of similarity measures with reduced dimensions [10, 28, 32]. The work in [6] is closely related to our work, in the sense that the authors propose an index structure that allows fast matching of time series, which are similar under scaling/shifting transforms. Our index structure requirements are different, as we store both signals and transformations and our similarity metric is based on $L_\infty$. Such constraints are not simultaneously handled in any prior work.

## 3. MULTI SENSOR DATA COMPRESSION

We use three real datasets in this paper to motivate and evaluate different techniques.

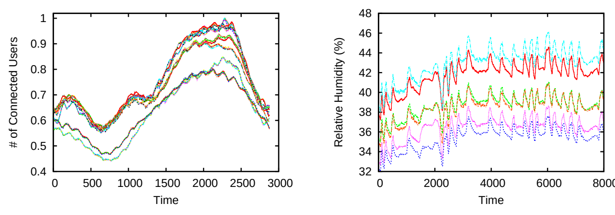
**DataCenter** dataset: This contains measurements of relative humidity by 24 sensors in a real data center over a period of two years [23].

**Intel-Berkeley Temperature (IBT)** dataset: This contains temperature measurements by 54 sensors deployed on a single floor of the Intel-Berkeley Research lab, over the period of one month [12].

**Server** Dataset: This is a 2-weeks long trace from 240 production servers of a large instant messaging service. The trace contains several performance counters such as number of connected users, CPU utilization, memory usage, etc.

In all three cases, each sensor reports a measurement every 30 seconds.

In this section, we formally introduce our multi-sensor compression problem. We start with a simple scenario where signals share data values with each other, although the sharing can change over time (Figure 1(a)). In Section 4, we will consider a more general scenario where signals differ significantly in values, but have similar forms (Figure 1(b)).



(a) Server data (b) DataCenter data

**Figure 1: Portions of two real-world datasets.**

### 3.1 Problem Formulation

Our general methodology can be summarized as follows. Given a set of $k$ sensors, with the $i$th sensor generating an ordered list of measurements $S_i = \{v^i(t),\ t = 1, 2, \ldots, n\}$, our scheme produces a set of *template signals* $\mathcal{L} = \{L_1, L_2, \ldots, L_\ell\}$ that are used as approximations for the sensor measurements. Each sensor's measurements are then partitioned into time intervals, so that for each interval a single template is used for approximation. Specifically, the compressed representation $\alpha_i$ for the $i$th sensor includes a partition of the time line $(t_0^i, t_1^i, \ldots, t_{m_i}^i)$, and a template $L_{j+1}^i$ for each (semi-open) interval $[t_j^i, t_{j+1}^i)$, where $L_{j+1}^i \in \mathcal{L}$. The interpretation is that the value of sensor $i$ during the time interval $[t_j^i, t_{j+1}^i)$ is approximated by the template signal $L_{j+1}^i$. For an arbitrary time instant $t$, we will slightly abuse the notation to write $\alpha_i(t)$ as shorthand for the value this approximation. (That is, if $t \in [t_j^i, t_{j+1}^i)$, then $\alpha_i(t)$ equals $L_{j+1}^i$ evaluated at $t$.)

Finally, suppose that template $L_i$ requires $m_i$ memory to store and is shared by $d_i$ signals, then our (compressed) representation for all the sensor signals requires memory

$$\sum_{i=1}^{\ell} (m_i + d_i)$$

The error of our approximation for $k$ sensors is

$$E(\mathcal{L}) = \max_{1 \le t \le n} \max_{1 \le i \le k} |v_i(t) - \alpha_i(t)|$$

which is the maximum difference at any time between the true signal and its approximation. Our optimization problem can be formulated as follows:

> *Given $S_1, S_2, \ldots, S_k$ and an error bound $\varepsilon$, choose template signals $\mathcal{L}$ with $E(\mathcal{L}) \le \varepsilon$ such that the memory required, $\sum_{i=1}^{\ell}(m_i + d_i)$, is as small as possible.*

At this level of generality, a good treatment of the problem remains elusive because arbitrary template signals are difficult to compute with and analyze. If, however, we approximate the signals with *piecewise constant functions*, then one can compute a polynomial-time approximation with a worst-case error bound. We call this the *interval sharing* scheme. Since piecewise constant approximation for single time series are popular, this result generalizes those approximations to multi-sensor data, taking advantage of periods when groups of sensors have similar values.

In the rest of the section, we present our *Interval Sharing Algorithm* (ISA) that, using the set cover heuristics, achieves a worst-case bound on the quality of approximation, given a memory budget for data representation. ISA accepts as

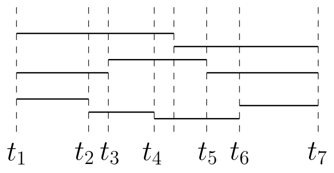


**Figure 2: 9 approximation segments corresponding to 3 different signals. These segments result in a division of the time-axis into 8 pieces.**

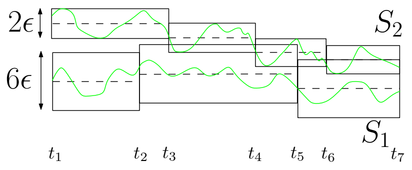


**Figure 3: Between time instants $t_3$ and $t_7$, the relaxed approximation segment bounding rectangles for signal $s_1$ intersects the approximation segment bounding rectangles of $s_2$.**

input a set $S = \{S_1, S_2, \ldots, S_k\}$ of signals and an error parameter $\varepsilon$. Our scheme works in two phases: the first phase approximates individual signals using piecewise constant curves; the second phase uses the well-known greedy set cover algorithm to share signal representation across sensors. For the first phase, we adopt a simple greedy bucketing scheme of Buragohain et al. [3]; also see [19]. For the sake of completeness, we briefly describe the bucketing scheme of [3] below since our set-covering phase both generalizes it and extends its error analysis to multiple signals.

## 3.2 Approximation of a Single Signal

Given a single time series and the maximum allowable relative error $\varepsilon$, the bucketing scheme of [3] greedily scans the series in order, inserting data items in the first bucket until the difference between the maximum and the minimum entries in the bucket exceeds $2\varepsilon$, at which point a new bucket is created. Since the gap between the maximum and the minimum entry in any bucket is less than $2\varepsilon$, all these entries can be approximated within the tolerance of $\varepsilon$ by the *average* of the max and the min. Viewed graphically, one can also see that all entries in a bucket are contained in an axis-parallel rectangle whose height is at most $2\varepsilon$. See Figure 3 for an illustration. It is shown in [3] that this greedy scheme uses an optimal number of buckets.

Now suppose that each of the $k$ signals in our set have been approximated using this scheme. The second phase of our algorithm attempts to achieve additional compression by sharing portions of buckets across different sensor series. It is instructive to take a geometric view. Each bucket in the individual series representation is a rectangular box. In order to share these boxes, we increase the error tolerance from $\varepsilon$ to $3\varepsilon$ and formulate a set-covering problem that attempts to find the smallest number of rectangular boxes that cover all the signals.

## 3.3 Compression by Interval Sharing

Suppose the bucket representation of signal $S_i$ has size $z_i$, meaning it uses $z_i$ rectangles. We partition the time axis into $z-1$ non-overlapping pieces, where $z = \sum_{1 \le i \le k} z_i$. See Figure 2 for illustration, where we show 8 segments corresponding to 3 signals and 7 pieces formed by the approximation segment endpoints. Next, for each series $S_i$, we construct $\binom{z-1}{2}$ sets, where a set corresponds to a possible subinterval of time for which we consider interval sharing. With each such interval, we associate a weight $w^i_{jl}$, which corresponds to the representation cost of signal $S_i$ in the interval between time instants $t_j$ and $t_l$.

We now explain how the weights are chosen. Please refer to Figure 3, which shows two signals, $S_1$ and $S_2$, and their approximations obtained by the bucketing algorithm. In addition to the bounding boxes for the approximation of $S_2$, we also show *relaxed* boxes for the $S_1$ approximation. A relaxed box is obtained by scaling a box along the $y$-axis by a factor of 3. (Since the original boxes obtained from the bucketing approximation have width at most $2\varepsilon$, the relaxed boxes have width at most $6\varepsilon$.) We say that a signal $S_i$ *fully* covers another signal $S_j$ in the time interval $(t_u, t_v)$ if the relaxed boxes of $S_i$ in this interval intersect *all* the bounding boxes of $S_j$ in that range. Intuitively, such an intersection implies that $S_j$ can be approximated by $S_i$ in this interval within $L_\infty$ error at most $5\varepsilon$.

The weight $w^i_{jl}$ is a fraction, where the numerator denotes the *representation cost* (memory) of $S_i$'s (bucket-based) approximation in the range $(t_j, t_l)$, and the denominator is the sum of these costs over all those series that $S_i$ *fully* covers in the range $(t_j, t_l)$. (Of course, the set corresponding to $w^i_{jl}$ is nothing but the collection of all the *pieces* of those time series that are fully covered by $S_i$ in the range $(t_j, t_l)$.) As an example consider the weight corresponding to range $(t_3, t_7)$ in Figure 3. If the memory needed to represent a contiguous set of $s$ segments is $2s+1$, then for the weight corresponding to $S_1$ in range $(t_3, t_7)$ the numerator is 5 (2*2 + 1) and the denominator is 12 ((2*2 + 1) + (2*3 + 1)) (partial segment is counted as 1).

We derive weights for all the sets in the manner described above. The universe consists of all the pieces and now we run a weighted set cover approximation algorithm [13] to get a solution to our problem. The set covering scheme proposed above achieves an $(\alpha, \beta)$ approximation, where $\alpha = 5$ and $\beta = O(\log k + \log(OPT))$. Due to lack of space, we omit a detailed analysis of this scheme, and simply summarize the main result.

THEOREM 1. *There is a polynomial time algorithm that achieves* $(5, O(\log k + \log(OPT))$ *approximation for representing $k$ time series using piecewise constant templates.*

PROOF. The proof for this can be found in the appendix. □

A similar result also holds for piecewise *linear* templates, although in our simulations and empirical evaluation we use piecewise constant templates due to their compact representation and ease of computation.

## 4. GAMPS: COMPRESSION BY GROUPING AND AMPLITUDE SCALING

Interval sharing is effective when multiple sensor's data have roughly the same values over periods of time. While some signals demonstrate this property (e.g., Figure 1(a)), many sensor signals from physical sensors do not share the same data value, yet demonstrate strong correlation in their patterns. For example, the data from 6 humidity sensors in Figure 1(b) are all different in values, but strongly correlated. In this section we show that our basic compression framework is still useful for such signals. Our general framework, called GAMPS, consists of two major components: *grouping*, which groups similar, but potentially non-overlapped, signals together, and *amplitude scaling*, which transforms signals in each group to a form suitable for template sharing. We describe these two components in the rest of the section.

### 4.1 Amplitude Scaling

One possible way to transform signals is to use linear regression. Suppose, two signals $S_i$ and $S_j$ are correlated and $S_i$ can be approximated as $S_i = a * S_j + b$, where the scalar coefficients $a$ and $b$ are determined to minimize a target error metric. Then, a compressed representation of $S_i$ can be reconstructed from a compressed representation of $S_j$ together with the coefficients $a$ and $b$. In practice, signals need to be segmented appropriately before parameters $a, b$ are determined for each segment. SBR [9] uses standard regression techniques to compute $a$ and $b$ to minimize the $L_2$ error metric. Calculating the $a, b$ parameters for our target $L_\infty$ metric is hard to accomplish—the solution is based on the well known Chebyshev approximation problem, which can be solved with a randomized linear programming algorithm. But this is an expensive operation and can provide only a probabilistic guarantee on $L_\infty$ error. Moreover, to support a small $L_\infty$ error bound, the parameters $a, b$ need to be computed on small segments of the signals, which reduces overall compression ratio and introduces additional computation load to determine the optimal segment size.

We address these difficulties by independently transforming individual data points in $S_j$, instead of using the same transformation parameters $a, b$ for the entire signal (or segment). More precisely, we use a *ratio signal* of $S_i$ and $S_j$. Let $S_i = \{v^i(t),\ t = 1, 2, \ldots, n\}$ and $S_j = \{v^j(t),\ t = 1, 2, \ldots, n\}$ be the signals of sensors $i$ and $j$. For the analysis shown in the rest of this paper, we assume that all signal values are positive, the equations for the signals with negative values can be obtained by using the absolute values. Let $\rho^i_j(t) = v^i(t)/v^j(t)$ be the ratio between the signals of $i$ and $j$ at time $t$. Then, we may define the *ratio* signal to be the time-ordered series of these ratios, namely,

$$R(S_i, S_j) = \{\rho^i_j(t),\ t = 1, 2, \ldots, n\},$$

which measures the *relative* values of $S_i$ with respect to $S_j$. That is, we interpret $S_j$ as a *base signal*, then the ratio terms tell us how to obtain corresponding terms of the signal $S_i$.

#### 4.1.1 *Using Ratio Signals*

In addition to, or in place of, the ratio signal mentioned above, one can imagine using a *delta signal*, where $\rho^i_j(t) = v^i(t) - v^j(t)$. Ideally, we would like these signals to be as flat as possible, so that they can be compressed using less space. The natural question, therefore, is this: *which signal is more compressible?*

Interestingly, we found that ratio signal is significantly more compressible than delta signal for many real-world sensor signals. For the ratio signal to be effective, it needs to be relatively flat: that is, at any time $t$, the ratio $v^i(t)/v^j(t)$ does not change much within a small window of time $\Delta$. A sufficient condition for this to hold is $(v^i(t + \Delta) - v^i(t)) \approx c(v^j(t+\Delta) - v^j(t))$, where $c = v^i(t)/v^j(t)$. In other words, a higher signal value changes more than a lower signal value, in approximately the same ratio of the signal values. This is generally the case for many *correlated* physical sensors, as we argue below.

**1.** When multiple sensors are deployed in the same physical environment, one sensor signal $S_i$ can be related to another signal $S_j$ according to a function $S_i = F(S_j)$. According to the Linear System theory, even if the function $F$ is non-linear, over a small time window $\Delta$, the change of the two signals can be linearly approximated well as $(S_i(t + \Delta) - S_i(t)) = c(S_j(t + \Delta) - S_j(t))$, for a constant $c$ dependent on $S_i(t)$ and $S_j(t)$ [30]. For such systems, a ratio signal can be very flat, since $S_j(t)/S_i(t) \approx S_j(t + \Delta)/S_i(t + \Delta)$, while a delta signal can be noisy. To make it more concrete, suppose in a data center, two temperature sensors deployed at two servers placed on top of each other report readings $S_i(t)$ and $S_j(t)$. The signal $S_j$ is coming from the top sensor and suppose that due to thermodynamics in the data center, $S_j(t) > S_i(t)$. Now, due to increased CPU load, both servers become hot resulting in $S_i(t + 1) = S_i(t) + \delta$. In addition, the heat dissipated by the lower server will move upwards (as hot air rises) to make the upper sensor's measurement $S_j(t + 1) = S_j(t) + \delta + f(S_i(t + 1))$. The function $f$ can be non-linear, and the difference $S_j(t + 1) - S_i(t + 1)$ may vary a lot, but according to the Linear System theory, the ratio $S_j(t+1)/S_i(t+1)$ will be approximately constant. The same is true in scenarios where sensors are deployed in an open space or in water and wind or water flow can push a physical phenomenon (e.g., heat) in one direction, causing some sensors to experience more cumulative effect than the others.

**2.** The ratio signal can be relatively flat even if the sensors do not directly affect each other. Some physical properties depend on another physical property in a non-linear way—for instance, the relative humidity depends on temperature in a non-linear way; we omit the physics behind this for lack of space. Consider two humidity sensors $i$ and $j$ deployed in two different locations at temperatures $t_i$ and $t_j$, respectively. Suppose temperature increases by a small value $\delta$. This will lead to different increases in relative humidity values at two sensors because the humidity increases at different rates at different temperatures. More specifically, $i$ will experience a larger increase in relative humidity compared to $j$, and the humidity ratio at the new temperatures $t_i + \delta$ and $t_j + \delta$ remains approximately the same as that at temperatures $t_i$ and $t_j$.

**3.** In many cases, sensing of non-physical phenomena also yields signals that obey the ratio rule. For example, consider the CPU loads of two servers (e.g., in the Server dataset) and assume that the first server has more connected users and therefore has a higher load than the second server. Now, suppose that in the morning all users in the system start increased activities, which will cause the first server to experience a higher additional load than the second server because of its larger user base. The delta between the loads of the two servers will change, but their ratio will remain approximately constant.

More generally, when signals change smoothly and their changes over a small window of time are very small compared to their absolute values, ratio signals are flatter than delta signals. To see this, suppose $v^i(t+\Delta) - v^i(t) = \delta^i(t)$ and $v^j(t+\Delta) - v^j(t) = \delta^j(t)$ and $\delta^i(t), \delta^j(t) \ll v^i(t), v^j(t)$. Let us denote the change in ratio signal as $\mathcal{D}^t_{ratio} = v^i(t+\Delta)/v^j(t+\Delta) - v^i(t)/v^j(t)$ and the change in delta signal as $\mathcal{D}^t_{delta} = (v^i(t+\Delta) - v^j(t+\Delta)) - (v^i(t) - v^j(t))$. It is easy to show that $\mathcal{D}^t_{ratio} \approx \mathcal{D}^t_{delta}/v^j(t)$. Thus, ratio signals have a much smaller amplitude range, and therefore admit more space-efficient representation for a given $L_\infty$ error $\varepsilon$.

In all the above scenarios, a ratio signal tends to remain more or less constant, thereby provides better compression ratio, than their delta signal. To confirm this empirically, we compressed our datasets using delta signals as well but found that the delta signals are so noisy that they make collective compression *worse* than just compressing individual signals. We therefore consider only the ratio signal in the rest of the paper. However, delta signal might be a good choice for some signals (especially when the sensors are independent and are not sensing the physical world), and our framework supports using a delta signal as well.

### 4.1.2 Compressing Base and Ratio Signals

Motivated by the above observation, we separately compress signal $S_j$ and the ratio signal $R(S_i, S_j)$ so that later we can reconstruct the signal $S_i$ from these compressed versions with a bound on the maximum error $\varepsilon$. We use the piecewise constant bucket approximation of the previous section to compress these two signals with maximum error bounds $\varepsilon_1$ and $\varepsilon_2$. However, the values of $\varepsilon_1$ and $\varepsilon_2$ must be carefully chosen to bound the maximum error of the reconstructed signal $S_i$ within $\varepsilon$. The following theorem shows relationship between $\varepsilon$, $\varepsilon_1$, and $\varepsilon_2$, and can help choosing the value of one parameter given the values of other two.

THEOREM 2. *Suppose a base signal $S_j$ has a piecewise constant approximation with $L_\infty$ error $\varepsilon_1$ and the ratio signal $R(S_i, S_j)$ has a similar approximation with error $\varepsilon_2$, then the signal $S_i$ can be reconstructed with $L_\infty$ error at most*

$$\varepsilon \;\leq\; c_1\varepsilon_1 + c_2\varepsilon_2 + \varepsilon_1\varepsilon_2,$$

$$\textit{where} \quad c_1 = \max_{1\leq t\leq n} \frac{v^i(t)}{v^j(t)}, \quad c_2 = \max_{1\leq t\leq n} v^j(t)$$

PROOF. The worst-case $\varepsilon$ is the product of the worst-case approximation error in the base signal and the worst-case error in the ratio signal. Consider the signal reconstruction $\hat{v}^i(t)$ of signal $S_i$ at time $t$. Then, we have

$$\hat{v}^i(t) \;\leq\; (\rho^i_j(t) + \varepsilon_2)(v^j(t) + \varepsilon_1) \tag{1}$$

But since the true value of the signal $v^i(t)$ satisfies $v^i(t) = \rho^i_j(t)v^j(t)$, we have the following

$$\hat{v}^i(t) \;=\; v^i(t) + \rho^i_j(t)\varepsilon_1 + \varepsilon_2 v^j(t) + \varepsilon_1\varepsilon_2, \tag{2}$$

which implies that

$$\hat{v}^i(t) \;-\; v^i(t) \;=\; \rho^i_j(t)\varepsilon_1 + \varepsilon_2 v^j(t) + \varepsilon_1\varepsilon_2 \tag{3}$$

The difference $\hat{v}^i(t) - v^i(t)$ is the approximation error, and so this completes the proof. □

Thus, the broad outline of GAMPS is the following. First we group the sensor signals into similar groups—this is the step we will describe in the next subsection—and then in each group we choose one signal as a base, and compute the ratio signals of the rest with respect to the base signal. In fact, the choice of the base signal is also important, and this is also a byproduct of our grouping phase. The key insight is that *the ratio signals of all the signals in a group are highly compressible,* and are well-suited for the bucketing approximation and the interval sharing scheme discussed earlier. In fact, in practice we found that the ratio signals were so sparse that no interval sharing was needed at all! In the following, we illustrate the Amplitude Scaling on the DataCenter data.

### 4.1.3 An Example

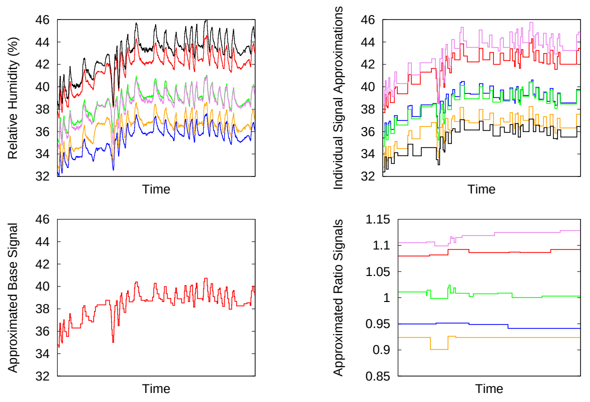


**Figure 4: The top row shows the humidity data and the result of individual signal compression. The bottom row shows the bucketing compression of a base signal, and the ratio signals of the other 5 signals. In all cases, the maximum allowed $L_\infty$ error is 1%.**

Consider six signals from the DataCenter data. Figure 4 (top-left) shows the original data, and (top-right) shows the result of applying bucketing approximation to individual sensor's data. The bottom two figures show the results of amplitude scaling on this data. The bottom-left shows the bucketing approximation of the base signal, using $\varepsilon_1 = 0.4\varepsilon$. The bottom-right shows the ratio signals of the other 5 sensors relative to the base signal. The ratio signals are also compressed and $\varepsilon_2$ is determined using Theorem 2. The reconstruction of any sensor data, using our amplitude scaling, results in $L_\infty$ error less than 1%, for any of the 6 sensors. approximations gives error less than 1% for all 6 signals. One can see that with amplitude scaling, we spend more memory representing a few base signals more accurately, but then most of the remaining signals can be represented very sparsely using their ratio signals.

### 4.1.4 Analysis of Amplitude Scaling

Our compressed representation of each sensor signal specifies both a template $L$, namely, the base signal, and a ratio signal $R$ for each time subinterval. Thus, in addition to the set of template signals $\mathcal{L} = \{L_1, L_2, \ldots, L_\ell\}$, we also have a set of ratio signals $\mathcal{R} = \{R_1, R_2, \ldots, R_{\ell'}\}$. The value of sensor $i$ during the time interval $[t^i_j, t^i_{j+1})$ is approximated by the template signal $L^i_{j+1}$ and the ratio signal $R^i_{j+1}$, where $R^i_{j+1} \in \mathcal{R}$ and $L^i_{j+1} \in \mathcal{L}$. Value of approximation $\alpha_i(t)$ (t $\in [t^i_j, t^i_{j+1})$) equals $L^i_{j+1}R^i_{j+1}$ evaluated at $t$. Suppose that the representation of template $L_i$ takes $m_i$ memory and is shared by $d_i$ signals, and the representation of ratio signal $R_i$ takes $r_i$ memory. Then the representation for all the sensor signals requires memory

$$\sum_{i=1}^{\ell}(m_i + d_i) + \sum_{i=1}^{\ell'} r_i$$

The error of our approximation is

$$E(\mathcal{L}, \mathcal{R}) = \max_{1 \le t \le n} \max_{1 \le i \le k} |v_i(t) - \alpha_i(t)|$$

Our *modified* optimization problem can now be formulated as follows: *given $S_1, S_2, \ldots, S_k$, and a error bound $\varepsilon$, choose the template signals $\mathcal{L}$ with $E(\mathcal{L}, \mathcal{R}) \le \varepsilon$ such that the memory required, $\sum_{i=1}^{\ell}(m_i + d_i) + \sum_{i=1}^{\ell'} r_i$, is as small as possible.*

For this problem as well, we can obtain a $(\alpha, \beta)$ approximation algorithm.

THEOREM 3. *There is a polynomial time algorithm that achieves $(3 + \triangle, O(\log n))$ approximation for representing k time series with piecewise constant ratio and template signals, where $\triangle$ is the worst case ratio of signal values at any time $t$, where $1 \le t \le n$.*

In fact, in the theorem above, $\triangle$ is the worst case ratio of two signals that are approximated together. We implement a variant of this algorithm, and for all datasets we used, this factor is well below 2. Our algorithm itself is a modification of interval sharing algorithm—it involves dynamic programming and solving (polynomially) many instances of a set cover variant. We omit all details about the algorithm and its proof due to space constraints.

While amplitude scaling can be combined with interval sharing, as mentioned in the theorem above, we found that a far superior combination was to use amplitude scaling together with a grouping scheme that groups the signals in a way to reduce the overall cost of templates and ratio signals.

## 4.2 Grouping of Sensor Signals

All given signals may not form a single group to be compressed together using amplitude scaling and interval sharing. This is not unexpected: spatial vicinity or other environmental factors may lead to similar trends among some small number of sensors, but any large facility would show significant variation across it. For instance, consider the IBT dataset, shown in Figure 5. While there are clearly common trends (and the ratio hypothesis holds), it is also clear that there are multiple distinct groups showing distinct trends. In order for our amplitude scaling to work well, we need an algorithm that can automatically discover a good grouping.

The goal of our grouping algorithm is different from existing time series clustering algorithms—we need to group signals together that can be compressed well with amplitude scaling and interval sharing. Clustering signals based on intuitive similarity measures can perform very badly in worst case scenarios. Consider two signals $S_1$ and $S_2$, where $S_2$ is a shifted version of $S_1$—in the first half $S_2 = 1.1S_1$

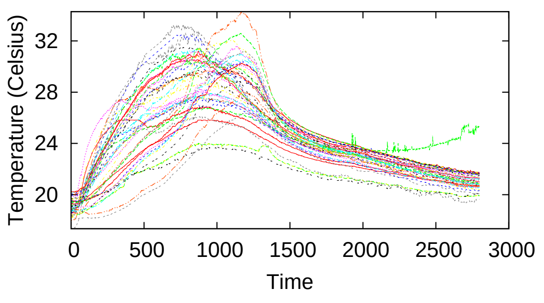


**Figure 5: A portion of the IBT dataset.**

and in the second half $S_2 = 0.9S_1$. Using similarity metrics such as Pearson's correlation coefficient $\rho$ will not put these two signals in the same group due to a very small value of $\rho$. However, when using amplitude scaling, it is easy to see that we get much bigger memory savings by grouping and compressing $S_1$ with $S_2$ together.

Fortunately, our cost metric allows us to formulate this as a *facility location problem*, giving us a worst-case bound on the approximation quality. We first describe a single-shot (static) grouping, and then extend it to the dynamic case where the grouping is allowed to change over time.

### 4.2.1 Static Grouping by Facility Location

Our goal is to identify a subset of the signals as base signals, and partition the remaining into groups so each signal is associated with a base signal. Each of these non-base signals will be represented by a ratio signal with respect to the base signal. Our optimization criterion, of course, is the total representation cost subject to the constraint that any signal can be reconstructed with $L_\infty$ error at most $\varepsilon$. There are two contributing factors in our representation cost: (1) the bucket approximation of each base signal, and (2) the bucket representation of the ratio signals of the remaining non-base signals. It turns out that we can formulate this as a classic facility location problem from combinatorial optimization [2, 18].

A general facility location problem is modeled as a graph $G(V, E)$, where a subset of the nodes $F \subseteq V$ is chosen as *facilities* (such as hospitals or fire stations) to service the remaining nodes. Opening a facility at a node $j$ incurs a non-negative cost $c(j)$. Servicing a node (client) $i$ using facility at node $j$ incurs a cost $w(i, j)$. (We assume that $w(i, i) = 0$, meaning that a facility serves itself for free.) The *facility location problem* is to find a set of facilities $F \subseteq V$ that can service *all* the clients with minimum possible total cost where we assume that each client is serviced by its closest facility. That is,

$$Minimize \quad \sum_{j \in F} c(j) + \sum_{i \in V} w(i, F)$$

where $w(i, F) = \min_{j \in F} w(i, j)$.

In our setting, the graph has the set of signals as the nodes, and the base signals are the facilities. Thus, $c(j)$, the cost of opening a facility at node $j$, equals the amount of memory required for representing an approximation of the base signal. The edge cost $w(i, j)$ in this graph, namely, the cost of servicing a client $i$ using facility $j$, is the memory

**Algorithm 1** StatGroup($\mathcal{S}$, $\varepsilon$)

```
1: $C = \emptyset$, $W = \emptyset$
2: $S' = S$
3: while $S' \neq \emptyset$ do
4:     Set $\varepsilon_1 = 0.4\varepsilon$ and determine $\varepsilon_2$ using $\varepsilon_1$.
5:     Pick one signal from $S'$, call it $S_j$.
6:     Let $c_j = Bucket(S_j, \varepsilon_1)$.
7:     $C = C \bigcup \{c_j\}$
8:     for all $S_i$ in $S$ do
9:         Take $S_j$ as the base signal, and compute ratio signal $R(S_i, S_j)$.
10:        Let $w(i,j) = Bucket(R(S_i, S_j), \varepsilon_2)$.
11:        $W = W \bigcup \{w(i,j)\}$
12:    end for
13:    $S' = S' \setminus \{S_i\}$
14: end while
15: Facility-Location(C, W)
16: Return the total cost of setting up the facilities and serving clients.
```

required to represent the ratio signal $R(S_i, S_j)$. The reduction is complete: minimizing the cost of the facility location problem minimizes the total memory needed to group the signals in such a way as to minimize the total representation cost for a given approximation error.

We should point out, however, that this is *not* a metric instance of the facility location, as the edge weights do not satisfy triangle inequality. Of course, the problem is NP-complete, but fortunately there is a known algorithm that yields $O(\log k)$ factor approximation, by reduction to weighted set cover, where $k$ is the number of signals.

Algorithm 1 gives a pseudo-code formulation of our static grouping scheme, called *StatGroup*, using the facility location algorithm as a subroutine. Let Bucket$(X, \varepsilon)$ denote a function which implements the bucketing algorithm, and returns the number of segments in the approximation. Algorithm 1 shows the StatGroup algorithm. Input to this algorithm is the set of signals $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ to be grouped and the desired maximum error $\varepsilon$. The output from the algorithm is the grouping and total memory consumed. The rest of the details are easily understandable from the algorithm.

### 4.2.2 Dynamic Grouping

The physical environment changes over time, and so a grouping of signals that is optimal at time $t$ may not be optimal in the future. Especially when archiving data over long durations of time, as in monitoring of large data centers for humidity and temperature, we expect trends to change, and so we need an algorithm that recomputes the groups when the old groups become suboptimal. With this in mind, we also implemented a dynamic version of our grouping scheme, shown in Algorithm 2.

The idea behind the Dynamic Grouping (DynGroup) heuristic is to start with a time window and compute an initial set of groups, using the StatGroup algorithm. We recompute the groups at each new window but we also *dynamically adjust* the window size to adapt to the underlying data. In each round, we can double the window size, halve it, or keep the same, depending on which size gives the best performance in terms of the memory use.

Here *wsize* denotes the current group size used for grouping the data. *ConstructSet*$(X, \tau_1, size)$ function constructs a subset of set $X$, starting from time instant $\tau_1$ and copying

**Algorithm 2** DynGroup(S, $\varepsilon$)

```
1: $wsize = 100$, $lastpt = 0$
2: while $lastpt \leq tsize$ do
3:     $S_{11} = ConstructSet(S, lastpt, wsize)$
4:     $S_{21} = ConstructSet(S, lastpt, wsize/2)$
5:     $S_{22} = ConstructSet(S, lastpt + wsize/2, wsize/2)$
6:     $m_{11} = StatGroup(S_{11}, \varepsilon)$, $m_{21} = StatGroup(S_{21}, \varepsilon)$, $m_{22} = StatGroup(S_{22}, \varepsilon)$
7:     if $m_{11} \geq m_{21} + m_{22}$ then
8:         $lastpt += wsize/2$
9:         $wsize = wsize/2$
10:    else
11:        $S_{12} = ConstructSet(S, lastpt + wsize, wsize)$
12:        $S_3 = ConstructSet(S, lastpt, wsize * 2)$
13:        $m_{12} = StatGroup(S_{12}, \varepsilon)$
14:        $m_3 = StatGroup(S_3, \varepsilon)$
15:        if $m_{11} + m_{12} \geq m_3$ then
16:            $lastpt += wsize * 2$
17:            $wsize = wsize * 2$
18:        else
19:            $lastpt += wsize$
20:        end if
21:    end if
22:    Update the index structure with the chosen representations in the last iteration.
23: end while
```

*size* samples of data (for all time series). To start off, the algorithm compares the memory taken by sticking with the current group size with the memory taken by halving the current group size. If there is an improvement by halving, the group size is halved. Otherwise, a memory comparison is done by doubling the current group size, and if this gives better memory savings group size is doubled, else it remains the same. The same proceedure is followed for every batch of data, with the batch size being the same as the current group size. The rest of algorithm is self-explanatory.

## 5. QUERYING COMPRESSED DATA

The primary goal of archiving sensor data is to be able to query it in future. An attractive aspect of our combinatorial compression is that the approximate representation of the signal lives in the original value space, and therefore lends itself to query processing, as opposed to schemes where the data must first be *decompressed* (e.g., schemes based on DFT). In this section, we describe a simple index structure, which we call *Multi Skip List* (MSL), which allows efficient processing of many generic queries. For illustrative purposes, we focus on the following basic queries.

- *Point Query*: Retrieve the value of the signal $S_i$ at time $t_j$. This is the basic recall query which GAMPS can answer within an error bound of $\varepsilon$.
- *Range Query*: Reconstruct the signal $S_i$ in the time period $[t_1, t_2]$, where $1 \leq t_1, t_2 \leq n$. In this case, we want a piecewise linear approximation of the signal for the period $[t, t']$ so that at no time does the approximation differ from the true value by more than $\varepsilon$.
- *Similarity Query*: Given two signals $S_i$ and $S_j$, find how similar they are within a time range $[t, t']$. Like [8], we use Jaccard similarity coefficient as the similarity metric.

Note that the output of a range query can be used by

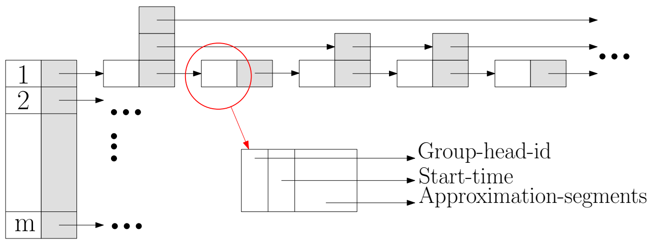


**Figure 6: The MSL Index structure.**

a variety of data mining queries. Our index structure is designed around the skip list, and is discussed below.

## 5.1 The Multi Skip List (MSL) Index

A pictorial representation of the index structure is shown in Figure 6. The first component of the MSL is the *Signal ID Index* (SI-Index) that maintains a mapping between the id of a signal and its compressed data. In Figure 6, SI-Index is organized as an array; however, if signals have non-numeric or non-consecutive IDs, SI-Index can be organized as a B-Tree, a hash table, or some other index structure that can efficiently look up the value for a given key (signal ID).

Each entry in the SI-Index points to a chain that represents the compressed data for the corresponding signal. Since compressed data is sequentially appended in GAMPS, data is naturally organized in sorted order of their timestamps. For efficient lookup based on timestamp, we organize each chain as a *skip list* [26]. A skip list is an ordered linked list with additional forward links, added in a randomized way with a geometric/negative binomial distribution, so that a search in the list may quickly skip parts of the list. In terms of efficiency, it is comparable to a binary search tree (O(log n) average time for most operations). In MSL, each skip list node has the following fields: (*Group-head-id*, *Start-time*, *Approximation-segments*). Start-time field contains the starting time of first approximation segment. Thus, a node with with Start-time $t$ contains compressed data within the time range $[t, t')$, where $t'$ is the Start-time of the next node in the skip list. Group-head-id field contains -1 if this signal itself is the group head within the interval $[t, t')$; otherwise it contains the ID of its base signal. Approximation-segments field either contains the segments approximating the signal itself (if Group-head-id field is -1) or it contains the segments approximating the ratio signal. Thus, if a node represents a base signal (if Group-head-id field is -1), it can be reconstructed by using its Approximation-segments; otherwise, it can be reconstructed by using the Approximation-segments of the base signal given by its Group-head-id, and its Approximation-segments representing the ratio signal.

Every time the grouping structure changes for any signal, a new node is created (the Group-head-id field reflects this grouping information change), inserted at the front of the corresponding skip list, and the SI-Index entry for the signal is updated accordingly. Thus, an insertion operation takes $O(1)$ time. Since no existing nodes (or their pointers) need to be modified, new nodes can be written on persistent storage sequentially. This leads to latency- and energy-efficient archival on magnetic disk and flash memory ([24] shows how insertion-only-on-front leads to efficient skip-list implementation on flash memory). If a node size must match the block size of persistent storage (for efficient write and read) and the node size reaches a block size, we start a new node, replicating the information in the Group-head-id field.

The final component of MSL is a second-level skip list embedded within each first-level skip list node discussed above. This second-level skip list is built on Approximation segments within a node so that searching within a node can be done in logarithmic time. This is particularly useful when a first-level skip list node contains a large number of Approximation segments over a large time window (which may happen if the grouping of a set of signals does not change, and hence signals are kept in the same node, for a long time).

## 5.2 Querying MSL Index

We now present how different queries mentioned before can be supported in MSL.

**Point query.** For a point query, we are given a signal-id, $S_i$, and time instant, $t_i$ and we need to find the value of $S_i$ at time $t_i$. To answer the query, we first search the skip-list of signal $S_i$ and find the node $N$ that contains the timestamp $t_i$. Then we search its Approximation-segments for the segment *seg* containing timestamp $t_i$. If the Group-head-id of the node $N$ is -1, then we return the value $v_1$ represented by segment *seg*. Otherwise, we repeat the above search in the skip-list of signal given by the Group-head-id of $N$, and find the value $v_2$ at $t_i$, and return the value $(v_2 * v_1)$ as the answer. Since all the searches for locating nodes and Approximation-segments are performed over skip lists, the overall time taken to answer this query is $O(\log n)$.

**Range query.** The range query $(t_1, t_2)$ can be answered in a similar manner by first searching for time instant $t_2$ and then following the skip list pointers till we find $t_1$ (data in a skip-list is stored in descending order of timestamps). The same process is repeated for the base signal if required. Overall, the query time is $O(\log n + \delta)$, where $\delta$ is the total number of Approximation-segments within the range $[t_1, t_2]$.

Other more application specific queries, like hot-spot queries for the sensor network signals, that asks for a histogram of the hottest sensors in a given time range, can be efficiently answered using these two basic queries.

**Similarity query.** After reconstructing signals by using the above range query, one can use any existing technique for measuring similarity between two signals. However, the grouping algorithm of GAMPS naturally and efficiently supports similarity queries based on signals' shapes (not absolute data values). Such a query is useful to identify signals that have similar trends, even though their absolute values can be different (e.g., Figure 1(b)). The key observation is that GAMPS groups two signals together if they have *similar* shapes within a time window represented by the group. Suppose, the function $b_{S_i,S_j}(t)$ returns 1 if two signals $S_i$ and $S_j$ are in the same group at time $t$, and 0 otherwise. Then, we define the similarity score $sim_{S_i,S_j}$ between signals $S_i$and $S_j$ within the range $[t_1, t_2]$ as follows:

$$sim_{S_i,S_j} = \frac{\sum_{t=t_1}^{t_2} b_{S_i,S_j}(t)}{t_2 - t_1} \tag{4}$$

Note that our similarity score is the same as the Jaccard similarity coefficient (for binary attributes $b_{S_i,S_j}(t)$), which has been shown effective to identify similarity and rarity of data streams [8].

To find the similarity score between two signals within a time range $[t_1, t_2]$ using MSL, we first find the first skip nodes (i.e., nodes containing the timestamp $t_2$) for both signals. This takes $O(\log n)$ time. Then, we sequentially scan both skip lists in parallel and accumulate the time over which both signals share the same group. Since the Group-head-id is defined once per node, this entire scan takes $O(max(k_1, k_2))$ time, where $k_1$ and $k_2$ are the numbers of skip nodes two signals have within the given time range. In contrast, had we first retrieved both signals and then used some linear time similarity algorithm, the total cost would have been $O(\log n + \delta_1 + \delta_2 + (t_2 - t_1))$, where $\delta_1$ and $\delta_2$ are the total number of Approximation-segments of two signals within the range $[t_1, t_2]$. Apart from the higher time complexity, this scheme might produce less accurate similarity results, as the similarity scores would be computed over approximate reconstruction of the original data; in contrast, our algorithm uses grouping information based on original signals. In Section 6, we will evaluate the effectiveness of our algorithm

Many other approximate forms of data analysis can be supported by GAMPS. For instance, the grouping information of GAMPS can be used to prune the search for anomaly detection. In particular, for detecting *magnitude* anomalies, we only need to look within a group; and for detecting *trend* anomalies, we need to only compare group heads of different groups. For more details on these anomalies we refer the reader to [20].

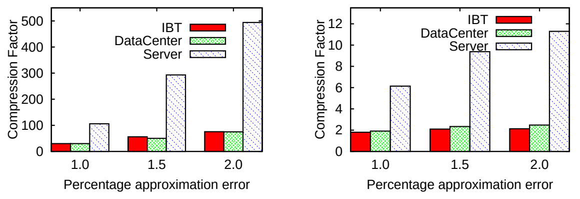
## 6. EXPERIMENTS

In this section, we report on our experimental evaluation of GAMPS, using 3 real-world data sets *DataCenter* (24 signals), *IBT* (54 signals), and *Server* (240 signals) described in the beginning of Section 3. We implemented the algorithm described in Algorithm 2. The facility location part (Facility-Location()) is formulated as a linear program (LP) and we use an open source LP solver for it. We evaluate GAMPS' compression performance, its dependence on error tolerance $\varepsilon$, the effect of grouping, and the accuracy of its similarity queries.

### 6.1 Compression Performance

Our first experiment confirms the obvious: compression can reduce space by several orders of magnitude. We ran GAMPS on all three data sets. Through all of our experiments, we divided the approximation error $\varepsilon$ into its two components by choosing $\varepsilon_1 = 0.4\varepsilon$, and then calculated the corresponding $\varepsilon_2$. We found that different choices of $\varepsilon_1$ had only a minor effect on the final compression, but one can certainly tune this parameter if needed. The results are shown in Figure 7(a). GAMPS reduces space by a factor between 100 and 490 on Server data, and between 25 to 125 for the other two datasets even for very small values of the approximation error $\varepsilon$, which is kept in the range [0.01, 0.02] (1–2%). The performance for the Server dataset is the best because it is the largest dataset and it has the largest average group size. For instance, with $\varepsilon = 1\%$, the average group size is 60 for the Server dataset, while it is only 4.5 and 6 for IBT and DataCenter datasets, respectively.

But *how good is GAMPS compared to some other approximation schemes?* We are not aware of any multi-sensor compression scheme that guarantees $L_\infty$ error, so we do the next best thing, and compare GAMPS to the state of the



(a) GAMPS vs. raw (b) Gamps vs. individual

**Figure 7: Compression obtained by GAMPS as compared to (a) raw data and (b) individually compressed signals.**

art *individual signal compression scheme*. That is, suppose we compress each signal $S_i$ individually to approximation error $\varepsilon$, how much additional compression can we expect from inter-signal correlation? The bucketing scheme of [3] is known to be optimal, so we use that to approximate individual signals, and compare the result with that of GAMPS. Pleasantly, we find that GAMPS is able to deliver space saving by a factor of 2 to 10 for an error of 1.5%, depending on the data (Figure 7(b)).

Figure 7 also shows the relative compression performance of GAMPS with increasing approximation error. Since increasing the error tolerance also improves the compression of individual signals, it is remarkable that on the Server data, the advantage of GAMPS continues to grow as $\varepsilon$ grows. By the very nature of the problem, compression across multiple sensors is only possible when those sensors show data correlation (which can vary dynamically over time). Thus, the performance of GAMPS is only significant when sensors can be grouped in large groups. We explore this aspect in the our next experiment.

### 6.2 Scaling with Group Size

In this experiment, we evaluate the effect of group size on the compression performance. We demonstrate this on the 240-signal Server data because that is our largest dataset. From this data, we extracted one group with 60 signals in order to evaluate the scaling of compression factor with increasing group size. Once again, we compare the performance of GAMPS relative to the best compression achieved by individual signal approximation using the bucketing scheme [3]. The results for this experiment are plotted in Figure 8. For this 60-signal Server data, GAMPS outperforms the individual signal compression by a factor between 8 when $\varepsilon = 1\%$ and 12 when $\varepsilon = 2\%$.

### 6.3 Compression Gain of Grouping

In this section, we empirically evaluate the significance of grouping in GAMPS by understanding how much space saving is attributable to grouping, to what extent a near-optimal grouping helps, and how much benefit comes from dynamic grouping.

#### 6.3.1 Static Grouping

In order to evaluate the importance of grouping, we first ran the simple-minded experiment where all the data is put in a single group. We then choose the best signal in the group as a base, to which the remaining signals are related by ratios. We observed uniformly poor performance because signals unrelated to the base signal lead to very bad ratio

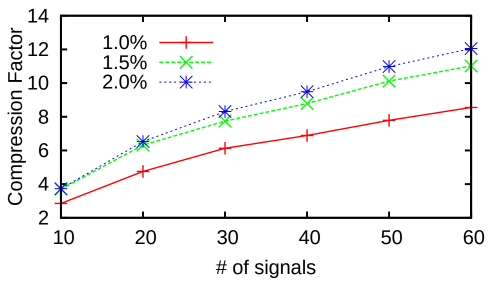


**Figure 8: Compression vs. the group size. The compression factor improves from 3 to 12 as the number of signals in the group varies from 10 to 60.**

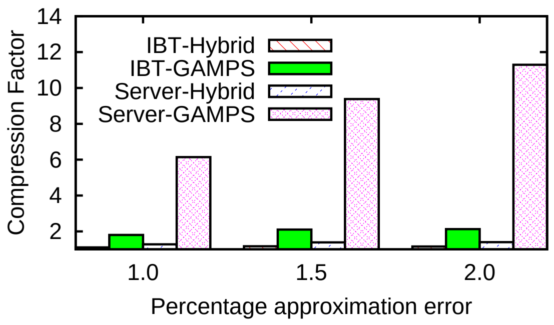


**Figure 9: Comparison between GAMPS and HYBRID IBT and Server datasets.**

signals. We, therefore, considered a slightly better version, which we call *hybrid*, where a single base is chosen for the group, but then the remaining signals have the freedom to either join the group (in which case we use its ratio signal) or maintain its own individual compression. The choice is made based on the cost of representation. We expect the hybrid to do at least as well as individual compression or a single group. Figure 9 shows the relative compression achieved by the individual compression, the hybrid, and GAMPS, using two data sets: the 45-signal IBT data and the 240-signal Server data. We note that a significant part of the compression factor for both the datasets, comes from the grouping. For instance, for the Server dataset for an error of 1.5% the compression factor achieved by the hybrid scheme over individual signal approximation is only 1.5 as compared to a factor 9 achieved by GAMPS.

One interesting thing to note is that the way the grouping algorithm is formulated, there is no information input to the grouping algorithm about the physical locality of the sensors. We only calculate the cost of representing one signal with another and input this to the grouping algorithm. Given all this, it is interesting to note the geographical locality of sensors placed in the same groups. Left part of Figure 10 shows the layout of the sensors in the Intel Berkeley lab, superimposed over the layout for the rooms. The sensors are shown by the dark hexagons in the figure. The part on the right shows the groups obtained geographically. The sensors in each of the shaded regions were placed in the same group by our grouping algorithm. The simplicity of

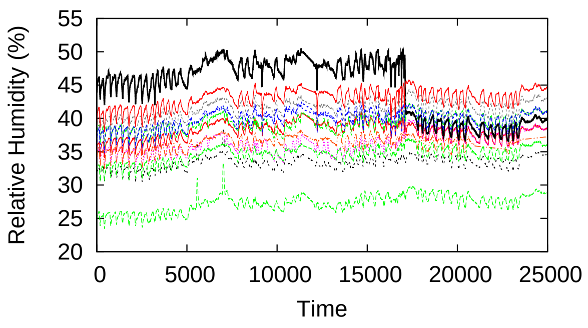


**Figure 11: Approximately 9 days of data for 12 sensors in the DataCenter dataset.**

the output group boundaries conforms to the intuition that sensors in the same group are likely to sense similar physical phenomena. Similar intuition has been presented by the authors of [12] who were behind the setup of this network. The authors in their work hand mapped a similar set of geographical regions in order to form an input for their model for prediction of sensor values, noting that the sensors in the same region, were expected to have similar sensed values. The sensors marked with crosses are the ones for which the data was not available for that day. The sensors marked with rectangles are *outliers*, i.e. the grouping algorithm used individual approximations for these sensors. Most of these outlier sensors are actually in or around conference rooms.

### 6.3.2 *Dynamic Grouping*

Next, we evaluate the power of dynamic grouping, relative to static grouping. Dynamic grouping has its tradeoffs. Every time we regroup, we need to store additional information about the new groups. There are also non-full buckets at the group boundary, which cause some waste of space. On the other hand, if the grouping interval is too long, we might not get the compression benefit which we can achieve by changing the grouping. In order to evaluate these pros and cons, we use the DataCenter data shown in Figure 11 for the experiment. Figure 12 shows our results on this dataset.

The key observation is that our dynamic grouping consistently achieves (albeit by a small margin) better performance than static grouping. Furthermore, our dynamic grouping scheme is able to adapt to data automatically in a way that is better than any of the fixed periodic regroupings we tried, with different frequencies (500, 1000, 2000 and 4000).

## 6.4 Signal Similarity Queries

Finally, we demonstrate the effectiveness of our similarity query algorithm using one example scenario; qualitatively similar results have been found with other scenarios. In this scenario, we use DataCenter dataset, a portion of which is shown in the top half of Figure 13. The bottom part of Figure 13 shows the output of most similar and most dissimilar signals for a query signal. Visually, the results look quite reasonable. As mentioned before, we are interested in similarity/dissimilarity in signals' shapes only. Therefore, ideally, one would compute correlation coefficients (e.g., Pearson's

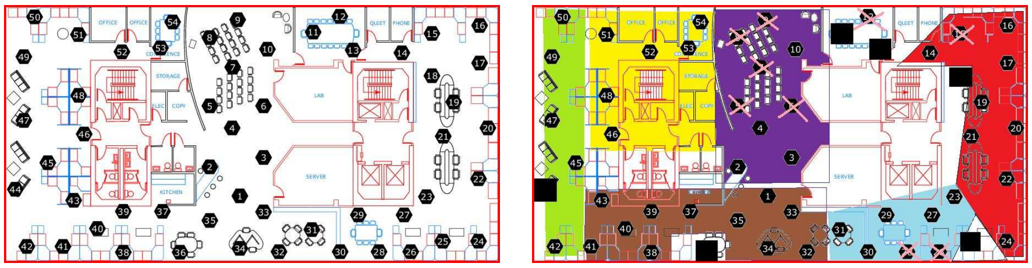


**Figure 10: The part on the left shows the sensor layout in the Intel Berkeley lab. The right part shows the groups found by the algorithm, along with outliers (marked by squares).**

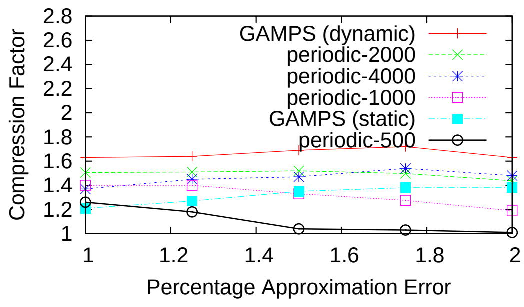


**Figure 12: Comparison between dynamic grouping (GAMPS), static grouping, and periodic grouping.**

coefficients $\rho$) on original signals to determine similarity—the higher the value of $\rho$, the higher the similarity of two signals' shapes. We found that the result of our scheme is very close to this ideal scheme; in particular, for the scenario in Figure 13, both our scheme and Pearson's coefficient based scheme determine the same signal as the most dissimilar ($\rho = 0.69$). For the most similar signal, the ideal scheme returns the most correlated signal with $\rho = 0.98$, while our scheme chooses the second most correlated signal with $\rho = 0.975$ which is very close to 0.98. In general, we found that our scheme may miss the most similar/dissimilar signal, but the $\rho$ values of the selected signals are very close to the true ones selected by the ideal scheme.

## 7. CONCLUSION

In this paper we have made a principled attempt to formalize the multi-sensor compression problem and proposed schemes with worst-case error guarantees. We have proposed GAMPS, a novel framework to compress a large number of sensor data streams. Unlike most existing work, GAMPS guarantees a given worst-case maximum ($L_\infty$) error. Notable features of GAMPS include i) it dynamically discovers groups of sensor signals that can be maximally compressed together, ii) it further improves overall compression ratio of signals using appropriate amplitude scaling, and iii) it maintains an index that enables answering several important queries directly from compressed data. To the best of our knowledge, GAMPS is the first system to provide these features. Evaluation of GAMPS with several real-world datasets shows significant benefit.

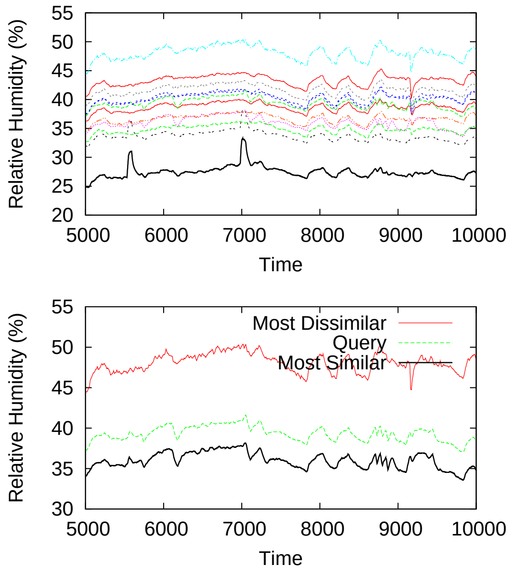


**Figure 13: Similarity queries.**

## 8. REFERENCES

[1] Agrawal, R., Faloutsos, C., and Swami, A. Efficient similarity search in sequence databases. In *FODO '93* (1993).

[2] Balinksi, M. L. On finding integer solutions to linear programs. In *IBM Scientific Computing Symposium on Combinatorial Problems* (1966), IBM, pp. 225–248.

[3] Buragohain, C., Shrivastava, N., and Suri, S. Space efficient streaming algorithms for the maximum error histogram. In *ICDE* (2007).

[4] Chan, K., and Fu, A. Efficient time series matching by wavelets. In *ICDE* (1999).

[5] Chen, Q., Chen, L., Lian, X., Liu, Y., and Yu, J. Indexable pla for efficient similarity search. In *VLDB* (2007).

[6] Chu, K., and Wong, M. Fast time-series searching with scaling and shifting. In *ACM PODS* (1999).

[7] Dang, T., Bulusu, N., and chi Feng, W. RIDA: A robust information-driven data compression architecture for irregular wireless sensor networks. In *EWSN* (2007).

[8] Datar, M., and Muthukrishnan, S. Estimating rarity and similarity over data stream windows. In *10th Annual European Symposium on Algorithms*

*(ESA)* (2002).

[9] Deligiannakis, A., Kotidis, Y., and Roussopoulos, N. Compressing historical information in sensor networks. In *ACM SIGMOD* (2004).

[10] Faloutsos, C., Ranganathan, M., and Manolopoulos, Y. Fast subsequence matching in time-series databases. In *ACM SIGMOD* (1994).

[11] Garofalakis, M., and Kumar, A. Deterministic wavelet thresholding for maximum-error metrics. In *ACM PODS* (2004).

[12] Guestrin, C., Bodik, P., Thibaux, R., Paskin, M., and Madden, S. Distributed regression: an efficient framework for modeling sensor network data. In *IPSN* (2004).

[13] Johnson, D. Approximation algorithms for combinatorial problems. In *ACM STOC* (1973).

[14] Keogh, E., Chakrabarti, K., Mehrotra, S., and Pazzani, M. Locally adaptive dimensionality reduction for indexing large time series databases. In *ACM SIGMOD* (2001).

[15] Keogh, E., and Pazzani, M. A simple dimensionality reduction technique for fast similarity search in large time series databases. In *PADKK* (2000).

[16] Kimura, N., and Latifi, S. A survey on data compression in wireless sensor networks. In *International Conference on Information Technology: Coding and Computing* (2005).

[17] Korn, F., Jagadish, H., and Faloutsos, C. Efficiently supporting ad hoc queries in large datasets of time sequences. In *ACM SIGMOD* (1997).

[18] Kuehn, A., and Hamburger, M. A heuristic program for locating warehouses. *Management Science 9* (1963), 643–666.

[19] Lazaridis, I., and Mehrotra, S. Capturing sensor-generated time series with quality guarantees. In *ICDE* (2003).

[20] Li, X., and Han, J. Mining approximate top-k subspace anomalies in multi-dimensional time-series data. In *VLDB* (2007).

[21] Lin, J., Keogh, E., Lonardi, S., and Chiu, B. A symbolic representation of time series, with implications for streaming algorithms. In *DMKD* (2003).

[22] Lin, J., Vlachos, M., Keogh, E., and Gunopulos, D. Iterative incremental clustering of time series. In *EDBT* (2004).

[23] Liu, J., Priyantha, B., Zhao, F., Liang, C. M., Wang, Q., and James, S. Towards discovering data center genome using sensor nets. In *HotEmNets* (2008).

[24] Nath, S., and Gibbons, P. B. Online maintenance of very large random samples. In *VLDB* (2008).

[25] Papadimitriou, S., Sun, J., and Faloutsos, C. Streaming pattern discovery in multiple time-series. In *VLDB* (2005).

[26] Pugh, W. Skip lists: a probabilistic alternative to balanced trees. *Commun. ACM 33*, 6 (1990), 668–676.

[27] Rafiei, D., and Mendelzon, A. Similarity-based queries for time series data. In *SIGMOD '97* (1997).

[28] Rafiei, D., and Mendelzon, A. Querying time series data based on similarity. *IEEE TKDE 12*, 5 (2000), 675–693.

[29] Ramaswamy, S., and Rose, K. Shared description fusion coding for storage and selective retrieval of correlated sources. In *Data Compression Conference* (2008).

[30] Rugh, W. J. *Linear System Theory.* Prentice Hall, 1995.

[31] Slepian, D., and Wolf, J. Noiseless coding of correlated information sources. *IEEE Trans. on Info. Theory 19* (1973).

[32] Vlachos, M., Hadjieleftheriou, M., Gunopulos, D., and Keogh, E. Indexing multidimensional time-series. *The VLDB Journal 15*, 1 (2006), 1–20.

[33] Xiong, Z., Liveris, A. D., and Cheng, S. Distributed source coding for sensor networks. *IEEE Signal Processing Magazine 21* (2004).

## APPENDIX

## A. PROOF OF THEOREM 1

Proof. Consider any signal $S_i$ and time instant $t$, the value of signal $S_i$ at time $t$ is given by $v_t^i$. Also, let the value of the approximation for this time instant for $S_i$ be $\alpha_i(t)$. Then we need to show the following

$$|\alpha_i(t) - v_t^i| \leq 5\varepsilon \tag{5}$$

Notice that at any time instant t, $v_t^i$ is either approximated by its own approximation segment formed in phase 1 (error $\leq \varepsilon$) or by the approximation segment of some other signal, say $S_j$. The bounding box of $S_j$ corresponding to the segment approximating $v_t^j$ has two values on its boundary for time t. Let us denote the value closer to $v_t^i$ by $b_t^j$. Then

$$|\alpha_i(t) - b_t^j| \leq 3\varepsilon \tag{6}$$

This is true because the width of the bounding box $\leq 6\varepsilon$ and $\alpha_i(t)$ is the same as $\alpha_j(t)$. Also, we know that the relaxed bounding box intersects the bounding box of $S_i$ approximating the value at time $t$. Hence

$$|b_t^j - v_t^i| \leq 2\varepsilon \tag{7}$$

Combining the results in equations 6 and 7, we get the result desired in equation 5.

We now argue that for a maximum error $\varepsilon$, if any two signals could share any part of their approximation, then this would be captured in one or more of the sets formed in phase 2 of the algorithm. In order for two points (from different signals) to have a common approximation segment, the distance between them *must* be less than $2\varepsilon$. If distance between two points is less than $2\varepsilon$, then the relaxed bounding box of one *must* intersect with the bounding box of the other. And every such intersection is captured in some set. And hence solving the set cover formulation optimally, would result in a solution which takes memory no more than the optimal piecewise constant shared representation with error $\varepsilon$ and has error no more than $5\varepsilon$ as shown above.

The weighted set cover problem is NP-hard. Let us put a bound on the total number of sets formed in phase 2, which will then give a bound on the memory taken by the approximate solution as compared to optimal. Let us denote the maximum number of approximation segments for any signal in phase 1 by $z_m$, i.e. $z_m = \max_{1 \leq i \leq k}$. Then, the optimal shared representation with error $\varepsilon$ cannot do better than $z_m$, so $z_m \leq OPT$. The total number of pieces formed in phase is $O(z_m^2)$, as none of the signals have more than $z_m$ segments in their approximation. Total number of sets formed per sensor is $O(z_m^4)$ and hence the total number of sets overall is $O(kz_m^4)$, which gives us the bound mentioned in the theorem. $\square$