Copy No. _____

Defence Research and Development Canada
Recherche et développement pour la défense Canada

DEFENCE R&D DÉFENSE

In-Memory Analysis of Maritime Data Sets

A Database System for Efficiently Processing Very Large Data Sets in Maritime Domain Awareness and Operational Research

D. E. Schaub

Defence R&D Canada – Atlantic

Technical Memorandum
DRDC Atlantic TM 2013-211
April 2014

Canada

This page intentionally left blank.

In-Memory Analysis of Maritime Data Sets

A Database System for Efficiently Processing Very Large Data Sets in Maritime Domain Awareness and Operational Research

D. E. Schaub

Defence Research and Development Canada – Atlantic
Technical Memorandum
DRDC Atlantic TM 2013-211
April 2014

Abstract

The present work develops an in-memory database system that enables efficient statistical calculations on extremely large sets of maritime data. The conventional approach of interspersing calculations with queries to relational databases incurs significant latencies when data retrieval patterns induce inefficient hard-disk operations. Mitigation through re-ordering data access is often feasible but at the expense of significant additional development involving practices in advanced computer programming. As random access of computer memory is several orders of magnitude faster than secondary storage access, it is highly desirable—when possible—to store databases exclusively in memory. Through bit-shifting, dynamic dictionary compression, and domain-specific data reduction techniques, the present work demonstrates a functional data storage and processing system capable of storing the entire Global Positional Warehouse in the memory of a personal computer. Data are made available to the user through a simple application programming interface that allows rapid development of efficient, large-scale statistical analysis programs.

Résumé

Le travail actuel vise à élaborer un système de bases de données en mémoire qui permet les calculs statistiques efficaces de très grands jeux de données maritimes. L'approche conventionnelle des calculs intercalés avec des requêtes aux bases de données relationnelles provoque d'importants délais lorsque les modèles d'extraction de données génèrent des opérations inefficaces sur le disque dur. L'atténuation par le réordonnancement de l'accès aux données est souvent possible, mais au détriment d'importants développements plus poussés, y compris des pratiques avancées liées à la programmation informatique. Étant donné que l'accès à la mémoire vive d'un ordinateur est infiniment plus rapide que l'accès au stockage secondaire, il est fortement recommandé, autant que possible, de stocker les bases de données dans la mémoire. Grâce au repositionnement des bits, à la compression dynamique de dictionnaires et aux techniques de réduction de données spécifiques au domaine, le travail actuel démontre un système fonctionnel de traitement et de stockage de données capable de stocker l'Entrepôt de données de localisation mondiale en entier dans la mémoire d'un ordinateur personnel. Les données sont rendues disponibles à l'utilisateur à l'aide d'une interface simple de programmation d'applications qui permet le développement rapide de programmes d'analyses statistiques efficaces à grande échelle.

This page intentionally left blank.

Executive summary

In-Memory Analysis of Maritime Data Sets

D. E. Schaub; DRDC Atlantic TM 2013-211; Defence Research and Development Canada – Atlantic; April 2014.

Background: Over the past decade, maritime domain awareness has been revolutionized by vast increases in data volumes brought forth by sweeping collection of self-reported vessel contact reports. While these new data sources have been applied—with great benefit—to the recognized maritime picture, recent efforts towards information utilization have broadened to include deeper analysis with the aim of better understanding information sources and designing new tools for maritime domain awareness. Historically, such analysis would entail the use of relational database systems that often lack the speed necessary for large-scale data analysis. The present work addresses this issue through the design and implementation of a geospatially-aware, in-memory data storage and retrieval system that both accelerates complex analyses of very large maritime data sets and obviates the need for familiarity with advanced programming techniques.

This work supports the DRDC Applied Research Project 06eo *Situational Information for Enabling Development of Northern Awareness (SEDNA)*. The following research was conducted by DRDC, Atlantic Research Centre over a 3 month period.

Results: This work demonstrates an algorithm capable of storing and rapidly processing very large sets of maritime data (including the entire Global Positional Warehouse) on a personal computer. It is expected that the software will scale favourably with the data volumes anticipated by the proposed revisions to the Global Positional Warehouse.

Significance: This work will improve the recognized maritime picture by simplifying challenging analysis on information sources used in operations centres. The software's straightforward interface enables a variety of users to perform fast statistical analysis and data mining on the complete set of Global Positional Warehouse data using existing personal computer systems. The software is of particular value in maritime domain awareness research and development, operational research and analysis, and real-time calculations in operational settings.

Future Plans: The software will be deployed in the course of performing a comprehensive assessment of information presently stored in the Global Positional Warehouse, with attention given to quality, consistency, and sensor coverage. Further analysis will encompass several areas of maritime domain awareness research, including the extraction of empirically-derived vessel motion models for the improvement of the recognized maritime picture and pattern analysis in support of anomaly detection.

Sommaire

In-Memory Analysis of Maritime Data Sets

D. E. Schaub ; DRDC Atlantic TM 2013-211 ; Recherche et développement pour la défense Canada – Atlantique ; avril 2014.

Contexte : Au cours des dix dernières années, la connaissance de la situation maritime a été révolutionnée en raison d'augmentations importantes de volumes de données générées par une collecte très étendue d'informations autosignalées sur la position des navires dans les comptes rendus de contact. Bien que de nouvelles sources de données aient été appliquées (avec d'excellents avantages) à la situation maritime reconnue, les efforts récemment déployés sur l'utilisation de l'information ont été élargis afin d'inclure des analyses plus poussées visant à mieux comprendre les sources d'information et à concevoir de nouveaux outils pour la connaissance de la situation. Par le passé, ces analyses demandaient l'utilisation de systèmes de bases de données relationnels qui, souvent, n'étaient pas assez rapides pour effectuer l'analyse de données à grande échelle. Le travail actuel traite de cette situation grâce à la conception et à la mise en place d'un système de stockage de données en mémoire sensible à la localisation géospatiale et d'un système d'extraction qui permettent d'accélérer les analyses complexes de grands jeux de données maritimes et d'éviter le besoin de familiarisation aux techniques avancées de programmation.

Le travail appuie le projet de recherches appliquées 06eo de RDDC, intitulé Situational Information for Enabling Development of Northern Awareness (SEDNA) [informations sur la situation pour permettre le développement des connaissances dans le Nord]. La recherche suivante a été menée par le Centre de recherches de l'Atlantique de RDDC pendant une période supérieure à trois mois.

Résultats principaux : Le travail démontre un algorithme capable de stocker et de traiter rapidement de très grands jeux de données maritimes (y compris l'ensemble de l'Entrepôt de données de localisation mondiale) à l'aide d'un ordinateur personnel. Il faut s'attendre à ce que le logiciel s'ajuste adéquatement aux volumes de données anticipées à la suite des révisions proposées à l'Entrepôt de données de localisation mondiale.

Portée des résultats : Le travail améliorera la situation maritime reconnue en simplifiant l'analyse difficile des sources d'informations utilisées dans les centres d'opérations. L'interface directe du logiciel permet à une variété d'utilisateurs d'effectuer rapidement des analyses statistiques et d'explorer les données de jeux complets de données de l'Entrepôt de données de localisation mondiale à l'aide d'ordinateurs personnels en place. Le logiciel représente une certaine valeur pour la recherche et le développement concernant la situation maritime reconnue, l'analyse et la recherche opérationnelles et les calculs en temps réel en contexte opérationnel.

Recherches futures : Le logiciel sera déployé pendant l'évaluation en profondeur de l'information qui est actuellement stockée dans l'Entrepôt de données de localisation mondiale, une attention particulière sera donnée à la qualité, l'uniformité et à la couverture des capteurs. Une autre analyse plus poussée comprendra plusieurs secteurs de recherche de la connaissance de la situation maritime, y compris l'extraction de modèles de mouvement de navire établis de faon empirique en vue de l'amélioration de la situation maritime reconnue et de l'analyse des tendances à l'appui de la détection d'anomalie.

Acknowledgements

The author would like to thank A.W. Isenor (DRDC, Atlantic Research Centre) for constructive conversations during the preparation of this manuscript and Scott Syms (MARLANT N6) for providing the datasets used in this report.

Table of contents

List of figures

1 Introduction

Like many areas in defence and security, maritime domain awareness (MDA) has been witness to enormous increases in information volumes. Over the last several years, new data sources—particularly self-reports in the form of AIS messages—have significantly reshaped the operational environment, necessitating not only fundamental changes to information systems, but adjustments to agency structures and operator work patterns as well. The ensuing challenges have inspired a wealth of MDA and operational research, with much effort directed towards reducing operator workload and developing systems that quickly bring relevant information to the operator's attention. As a result of these changes, operations centres are able to monitor and respond to global maritime activities to a degree that is without precedent.

The collection and archival of vast sets of maritime data also provides opportunity for research in pursuit of objectives beyond immediate operational concerns. In the context of operational research (OR), such analysis might include evaluating measures of effectiveness based on sensor and fusion performance, recognized maritime picture quality, and Intelligence Surveillance Reconnaissance (ISR) metrics [1]. The results of these evaluations may then be used to effectively direct resources to improve the performance of existing infrastructure (or operations) and/or evaluate prospective sensing platforms and fusion capability. In the case of basic MDA research, a large corpus of vessel tracks may be used to extract statistical models for vessel motion, which may in turn be used to improve vessel tracking and target identification. The large datasets may also be used for pattern analysis in support of developing methods of anomaly detection and establishing vessel patterns of life, topics that may lead to systems that provide early warning to maritime emergencies or security-related issues.

The computational challenges in MDA research are substantially different from those encountered in operational settings. In the latter case, information systems are tailored towards real-time picture compilation and presentation that heavily emphasize the recency of information. As the utilization of archived data is limited, processing requirements are quite modest (by contemporary standards). In contrast, data sets in MDA research may be significantly larger and processing considerably more complex. Moreover, severe performance degradation may result when executing complicated calculations on data sets exceeding system memory. In this case, certain analyses may become infeasible when computations induce disk-access patterns that are highly inefficient. Re-ordering calculations may yield (more efficient) sequential disk access but usually incurs considerably greater development effort that shifts emphasis from the analysis problem itself to optimization techniques in computer programming. These issues, which frustrate efforts to expediently carry out analysis, may be avoided by ensuring adequate system memory, employing data compression, or both.

This work describes the development of a high-compression in-memory database system

that facilitates rapid statistical analysis of large maritime data sets and obviates the need for advanced programming techniques. The remainder of this report begins with theoretical considerations of data storage and retrieval in relation to different forms of computer memory. Information sources particular to MDA are then studied in the process of deriving in-memory data compression that supports efficient random-access. A fast sorting algorithm and geographical extensions are then derived. The report concludes with the demonstration of a user-defined analysis tool that employs the database system to very rapidly classify hundreds of millions of contact reports according to the ocean of origin. It is expected that the software's simple application programming interface and geographical extensions will enable the development and execution of a variety of additional complex analyses, particularly in maritime domain awareness research and development, operational research and analysis, and real-time calculations in operational settings.

2 Storage and Retrieval of Digital Data in Scientific Computing

This section provides a brief overview of computer memory theory in relation to data storage in digital systems. As these subjects are enormous in scope and depth, only aspects germane to the processing of large-scale data sets are covered. Readers desiring a comprehensive background are referred to [2].

Computer memory may be classified as either *primary* or *secondary* storage. The former (hereafter referenced simply as 'memory') is nearly always semiconductor-based, expensive (per unit of capacity), volatile (data are lost when powered-down), and extremely fast. The latter, which includes magnetic media such as hard drives, is relatively inexpensive and non-volatile, but exhibits considerably greater access latency and reduced bandwidth. The architecture of a typical computer system includes both forms of storage, wherein semiconductor-based memory—mostly random-access memory (RAM)—is tightly integrated and partially embedded with processing units, while secondary storage—usually in the form of hard disks—is connected to the central processing logic through peripheral buses and special controller hardware. While the instructions and data used in program execution normally reside in memory (an absolute requirement for imminently-executed instructions), operating systems may recruit certain portions of a system's hard-disk to serve as *virtual memory* in the event that primary storage becomes exhausted. In this case, the operating system transparently transfers data between memory and disk in a process known as *paging*.

Performance considerations in scientific computing often concern the tradeoff between memory speed and size (i.e. choosing between slower hard-disk storage and lower-capacity RAM) and latencies characteristic to different data access patterns. In particular, sequential reads or writes to consecutive physical locations are faster than their random counterparts. This phenomenon especially affects rotating media, where non-consecutive operations experience delays of several milliseconds that result from the physical repositioning of internal read/write assemblies. While the latencies incurred by isolated random accesses are of little significance, they can severely impact virtual-memory operations. These differences in speed are illustrated in Fig. 1, which shows the various times required to initiate the retrieval of a single byte of data on a typical personal computer.

A simplified workflow in scientific computing includes loading a file from disk into memory, processing data by performing numerical and/or logical calculations, and saving the results to disk upon completion. In this model of computation, disk access is seldom regarded as a significant performance inhibitor, as it occurs only during load and save operations that are completed quickly by way of sequential disk access. When data-set size exceeds a system's memory, this workflow must be adjusted to reflect that calculations can no longer be performed in memory alone; program execution necessarily effects hard-disk access, ei-

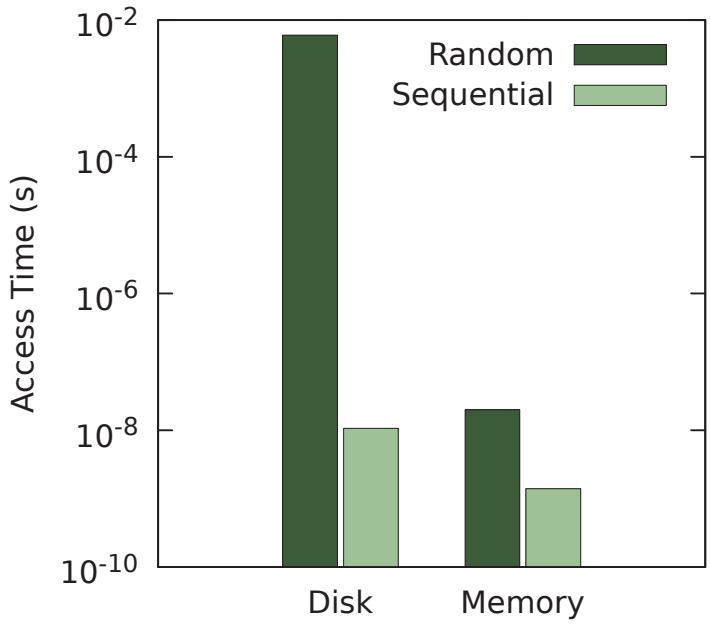

Figure 1: *Average single-byte retrieval speeds (in seconds) as measured on a Dell Precision M4600 personal computer. Note the ∼6 order-of-magnitude difference between random (magnetic) disk and (dynamic) memory access speeds.*

ther through paging mediated by the operating system, or by direct disk operations initiated by the software, and the execution speed becomes strongly influenced by the pattern and volume of hard-drive reads/writes. In the worst case, excessive paging (known as *thrashing*) will bring execution to a near halt and render infeasible normal program completion. Thrashing may be avoided through software that uses a minimum of carefully-designed sequential disk-memory transfers to reduce memory footprint and thereby eliminate reliance on virtual memory.[1] However, such optimizations part with conventional memory usage paradigms, requiring both proficiency in advanced concepts of computer science and a deeper insight into the computational nature of a given problem.

The foregoing arguments strongly advocate—whenever possible—memory-exclusive data processing. While other performance obstacles (such as those related to computational complexity) remain, in-memory processing allows for the simplest implementations of complex analyses and eliminates the potential for time-intensive optimizations, allowing attention to remain directed toward the domain-specific (as opposed to computational) aspects of analysis.

[1] Finite-element solvers such as Ansoft HFSS commonly employ this strategy.

3 Software Design

The software may be functionally decomposed into low-level compression/storage and decompression/retrieval operations, database initialization, high-level data access by the user-defined application, and ancillary functions that include sorting and geographical extensions. The underlying software architecture (Fig. 2) conforms the client-server design pattern; the server process reads data into a region of memory that is then shared with one or more user-defined client programs. The choice of shared memory in lieu of a standard interface (such as a sockets connection) minimizes data access latency but also exposes the database to inadvertent corruption by ill-defined user programs. This risk, however, may be avoided by using only the client functions defined by the software.

In the present work, the software (which is of a general nature) has been adapted to data sets exported from the Global Positional Warehouse (GPW), an information system maintained by MARLANT N6. This system, which archives millions of current and historical vessel tracks and contact reports from around the world, is central to the compilation of the recognized maritime picture used by the Regional Joint Operations Centres (RJOCs) and Marine Security Operations Centres (MSOCs). In addition to contact location, GPW also stores attributes such as a vessel's International Maritime Organization (IMO) Number, Maritime Mobile Service Identity (MMSI), name, call sign, flag, etc. Data was transferred from GPW to the software through sets of comma-separated-variable (CSV) files.

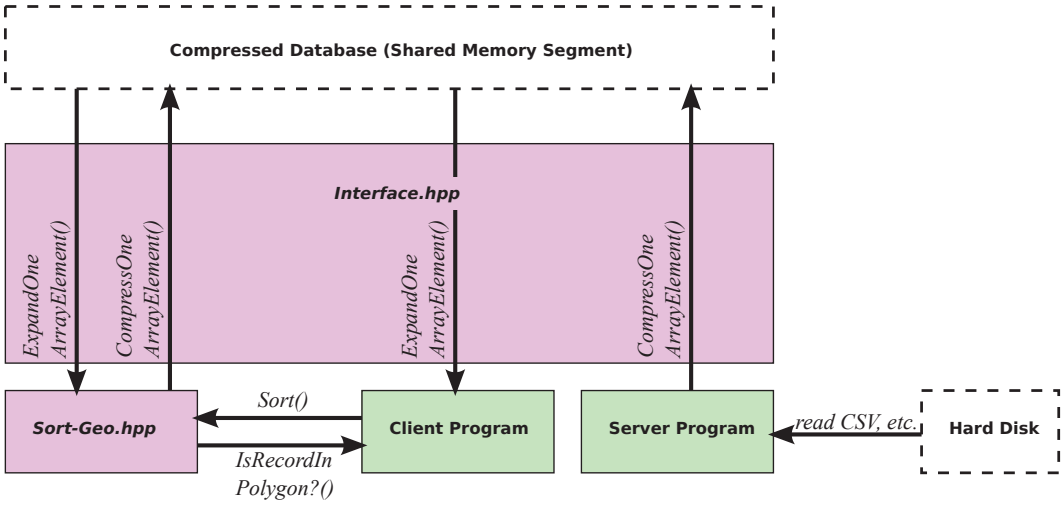

Figure 2: *Software architecture, where arrow orientation indicates direction of data flow, and boxes with dashed outlines represent physical storage. Note that data operations are always initiated by either the client or server programs.*

3.1 Compression and Decompression

The compression techniques employed are threefold: First, the absolute minimum space for a given field is derived in relation to domain-specific requirements. For example, latitude granularity better than 1 meter is practically unnecessary in view of the considerably larger positional error accompanying most contact reports. As the earth's circumference is ∼40,000 km, the latitude field need resolve only ∼40,000,000 discrete values, which may be stored in $\log_2(4\text{x}10^7) = 25.25$ bits. This process is repeated for similar fields such as depth, altitude, length, bearing, etc., and the resultant data sizes are rounded up to the nearest integer bit.

The second form of compression targets repetition in fields such as vessel name, flag, and MMSI and IMO identifiers. The resultant redundancy may be eliminated by substituting each repeated value for a unique index of considerably shorter length and defining a dictionary that associates indexes to field values. The logarithm of the resulting maximum index yields the space required to store the index in the record, which may be substantially less than that of the original data. This technique is known as *dictionary compression* and forms the basis of many compression algorithms such as LZW [3]. In most cases, further economies in data storage size—particularly for memory intensive-fields such as vessel name—may be realized by constructing minimum dictionaries that span only the range of *observed* data. Such an approach entails dynamically defining dictionaries during database initialization, a process that is described in the following section.

In most cases, compression is maximized by way of variable-length coding; data occurring with high frequency are represented by dictionary indices of shorter length and vice versa. While suited to applications that carry out linear passes on compressed data, variable-length coding impedes random access, as finding a given datum's memory address involves processing all preceding data. Variable length codes also preclude in-place sorting and sorting algorithms based on monotone-mapping dictionaries (§3.4). Strategies for circumventing these problems may be found, though at the cost of additional memory, complexity, and random-access latency. In contrast, fixed-length coding places no restrictions on sorting and allows memory addresses to be expressed as simple arithmetic functions of a datum's index (Fig. 3). For these reasons, variable-length dictionary encoding was not pursued.

The third element of compression involves eliminating compiler-generated memory gaps between adjacent fields and records. This unused space, which clearly results when native data types store fields of fewer bits, also occurs when compilers align variables to permissible memory addresses, required by conventional microprocessor design to be divisible by variable size. For example, an IEEE-754 double floating point variable (8 bytes) may be stored at addresses 16, 24 or 32, but not 6 or 7). Should the preceding variable definition be a single character (one byte), up to three bytes become unusable. The software completely eliminates such unused space by foregoing compiler-generated variable definitions and storing both individual fields and records in immediate succession (Fig. 4). In this con-

figuration, the memory address of a particular record (or field) must be expressed in bits.

Reading fields in the bit-mapped database (Fig. 4) requires an interface function *ExpandOneArrayElement*() that restores data to original—albeit still dictionary-compressed—data types. This operation involves copying the bytes containing the field into a temporary variable, zeroing the leading and trailing bits that were collaterally copied (belonging to adjacent fields in the database), and finally bit-shifting the temporary variable to recover the original data alignment. The contents of the temporary variable are then copied to the destination variable that is of the original data type. Storage of data to the database is using *CompressOneArrayElement*() occurs by way of the reverse process.

The reading/writing procedures make use of data-specific record layout sequences that contain the length (in bits) of each field in the order in which it appears in a record. For example, the sequence for a three-field record (26-bit latitude, 26-bit longitude, 5-bit identification number) would be the array [26, 26, 5].

3.2 Database Initialization

The in-memory database is constructed by the server program in two phases from a group of CSV files stored on disk (in the present work these files contain track and contact information from GPW). The first step consists of sequentially reading the entire set of files to compute the required memory and compile the sets of unique values to be defined in the field-specific dictionaries. If memory requirements are found to exceed the system memory, the program immediately terminates with error. The dictionary entries are tabulated using the *std::set* data type from the C++ standard template library [4]. By design, this data type maintains a sorted list of unique values, guaranteeing logarithmic (fast) insertion complexity and transparently handling duplicate insertions. Owing to the data structure's inter-

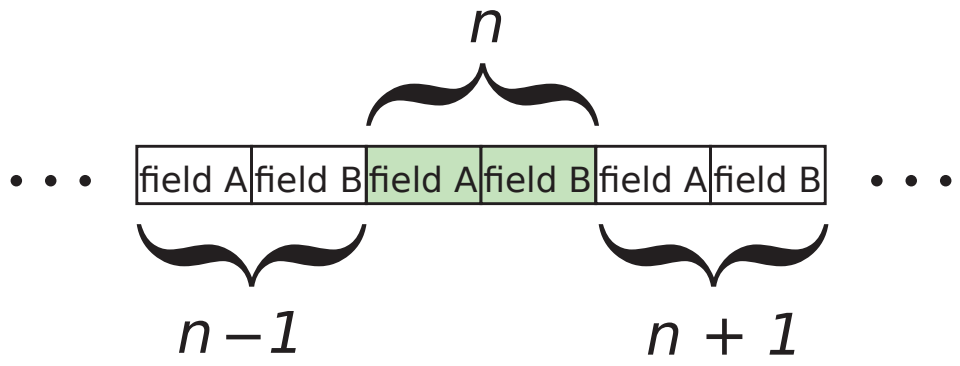

Figure 3: *Memory data map showing that records (indexed by n) are of uniform width and stored linearly in memory. The memory address of any given record can therefore be computed arithmetically. Note that records are not word (or even byte)-aligned.*

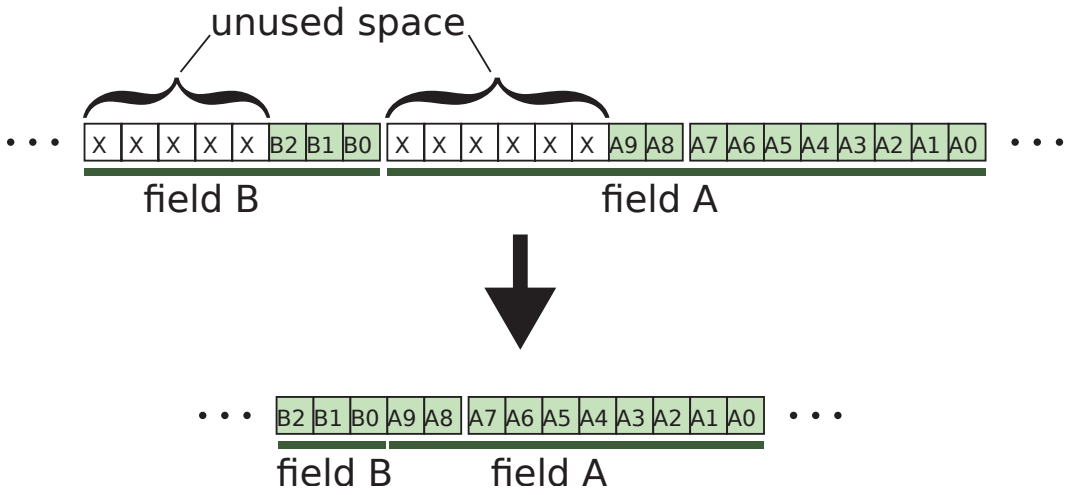

Figure 4: *Reduction of data size by reclaiming unused memory.*

nal order, the final set may be transferred directly to the respective monotone dictionary. Upon completion of the first step, the required memory is requested from the operating system as a single, contiguous block. The entire data set is then read from disk, compressed, and stored to memory, which is shared with client programs using the Boost.Interprocess library [5]. The shared memory segment remains available to client process for the duration of the server program's lifetime.

3.3 Database Access

Data may be retrieved and set through a collection of objects (instantiated C++ classes). The shared memory segment is accessed by user-defined programs through a pointer of type *GPWData* defined in the header file *Interface.hpp*. In turn, this data structure contains the set of pointers defined in Table 1. In each case, relative pointer arithmetic is used (Boost.Interprocess library) owing to distinct address spaces of the server and client programs.

Each of the contact and track pointers may be dereferenced to an object accessible by way of a C-style array operator $[\cdot]$. This (C++ overloaded) operator serves as a function that transparently reads (writes) and decompresses (compresses) single elements from the database, concealing complex bit-shifting operations (§3.1) from the user. Individual fields may be accessed through *get* and *set*-style member functions, which are defined in lieu of a second index to improve software legibility. For example, reading the MMSI of the 589^{th} record in the contacts database may be done as `data->(*ContactsPtr)[589].getMMSI()`.

Finally, each of the database fields is associated with a unique C++ class that is used

Table 1: *Data Pointers stored in the* GPWData *class.*

Pointer Name	Description
ContactsPtr	Pointer to the set of contacts
TracksPtr	Pointer to the set of tracks
MMSITable	Dictionary of MMSI values
IMOTable	Dictionary of IMO values
NameTable	Dictionary of vessel names
RadioTable	Dictionary of radio call signs
ClassTable	Dictionary of vessel class
SourceTable	Dictionary of source
SensorTable	Dictionary of sensor type
TypeOfTrackTable	Dictionary of track type

in place of native data types. This is done to facilitate error checking and ensure that only permissible values are stored to database fields. An illegal operation (such as `data->(*ContactsPtr)[589].setLatitude() = 700°`) results in a software exception being thrown. The server program catches exceptions during construction of the database, and in each case, displays an error message notifying the user to the invalid value that was encountered in the source data. Uncaught exceptions in user-defined programs will result in program termination, as it was deemed that invalid values could undermine the results of analysis.

3.4 Sort Operations

Although the software facilitates efficient random access, performance may further improve with cache optimization techniques. In the case of a single-threaded user-defined program, this amounts to accessing records in sequence. Fortunately, data analysis can often be derived in terms of sequential access by first using a sorting algorithm to reorder records in the database. Sorting is also required for using fast $O(\log n)$-complexity binary search algorithms.

Analysis of computer sorting may be profitably divided into a study of the underlying sort algorithm (which may be defined abstractly in terms of comparison operators that induce a partial order on a set of generalized records), and analysis of the realized comparison and memory operations used in a given implementation. In most applications, a fast algorithm such as heapsort, mergesort, or quicksort is used. These algorithms exhibit a common (expected) time complexity of $O(n \log n)$ but otherwise possess distinct performance profiles, particularly in regard to temporary-memory requirements and suitability for parallelization. Mergesort is often preferred for its speed, robustness (deterministic runtime), and ease with which cache-optimized and parallel variants may be implemented. However, as mergesort

requires (by design) significant additional memory, the software's sorting functions were instead based on an in-place—albeit slower[2]—heapsort algorithm.

A straightforward sorting implementation carries out comparisons between decompressed fields values, a process that is conceptually simple but incurs the modest overhead of repeatedly calling *ExpandOneArrayElement()* and carrying out dictionary lookups. In most cases, the mapping between dictionary index and respective definition is readily made *monotonic*, i.e. order preserving (Fig. 5). By shifting the compressed sort field (also known as the key) to a byte boundary, comparisons may be then be performed on the indices themselves by way of fast, machine-native integer operations. This process can be further optimized by padding record length to 8 bits and reinstating bit shifting as pre- and post-sorting operations. In this approach, the internal record layout is modified by moving the key to the beginning or end of each record, depending on processor endianness.[3] Upon completion of the main sort algorithm, this process is reversed in a manner that restores the original field order.

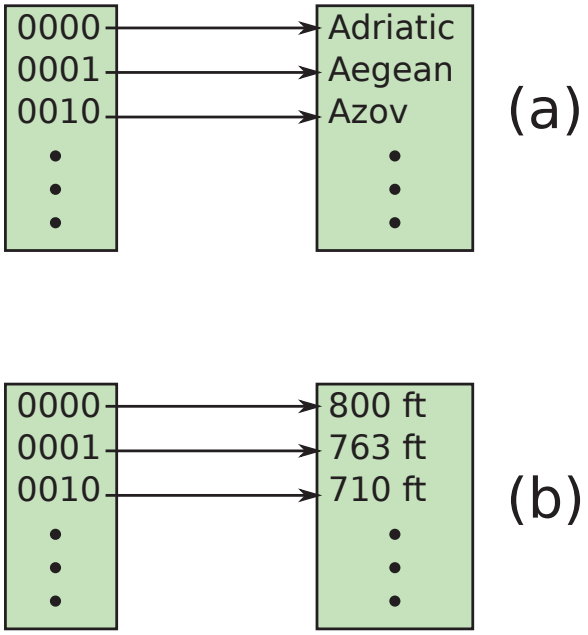

Figure 5: *Monotone mapping between dictionary and index for (a) vessel name and (b) vessel length (shown in reverse order). Such mappings preserve lexicographic order that is required by sorting operations.*

The foregoing optimization readily accommodates multi-column sorting[4] by suitable rearrangement of internal record structure that places sort fields in descending order of signif-

[2]This implementation is not cache optimized and cannot be parallelized.
[3]The SPARC and x86-64 architectures are big and little endian, respectively.
[4]E.g. a personnel database sorted first by surname and then by given name.

icance. As the resultant integer comparisons may span multiple fields, the sort direction (ascending or descending) must be uniform across keys, a requirement that may be satisfied by replacing fields to be sorted in the (arbitrarily chosen) contrary direction by their two's complements. This process is illustrated in Fig. 6 and summarized in pseudocode in Algorithm 1.

```
Data: DataArray,
      SortFields = {First Column, Second Column, ...},
      SortFieldsDirection = {ASC/DESC, ASC/DESC, ...}
Result: DataArray (sorted)
Initialization;
/* Step 1:  Process data to allow native comparison     */
for i ← 1 to length(DataArray) do
    tmp[ ] ← ExpandOneArrayElement (DataArray[i]);
    for j ← 1 to numColumns do
        tmp2[j] ← tmp[SortFields[j]];
        if SortFieldsDirection[j] = DESC then
            tmp2[j] ← Invert (tmp2[j])
        end
    end
    DataArray[i] ← CompressOneArrayElement (tmp2[ ]);
end
/* Step 2:  Build heap with in-place swaps              */
/* Smallest element on top                              */
BuildHeap;
for i ← length(DataArray) to 2 step -1 do
    /* Swap top of heap with element at bottom of list  */
    Swap (DataArray[0], DataArray[i]);
    RepairHeap;
end
/* Step 3:  Restore data to original state              */
/* (undo step 2)                                        */
for i ← 1 to length(DataArray) do
    tmp[ ] ← ExpandOneArrayElement (DataArray[i]);
    for j ← 1 to numColumns do
        tmp2[SortFields[j]] ← tmp[j];
        if SortFieldsDirection[SortFields[j]] = DESC then
            tmp2[SortFields[j]] ← Invert (tmp2[SortFields[j]])
        end
    end
    DataArray[i] ← CompressOneArrayElement (tmp2[ ]);
end
```

Algorithm 1: In-place heapsort for compressed data

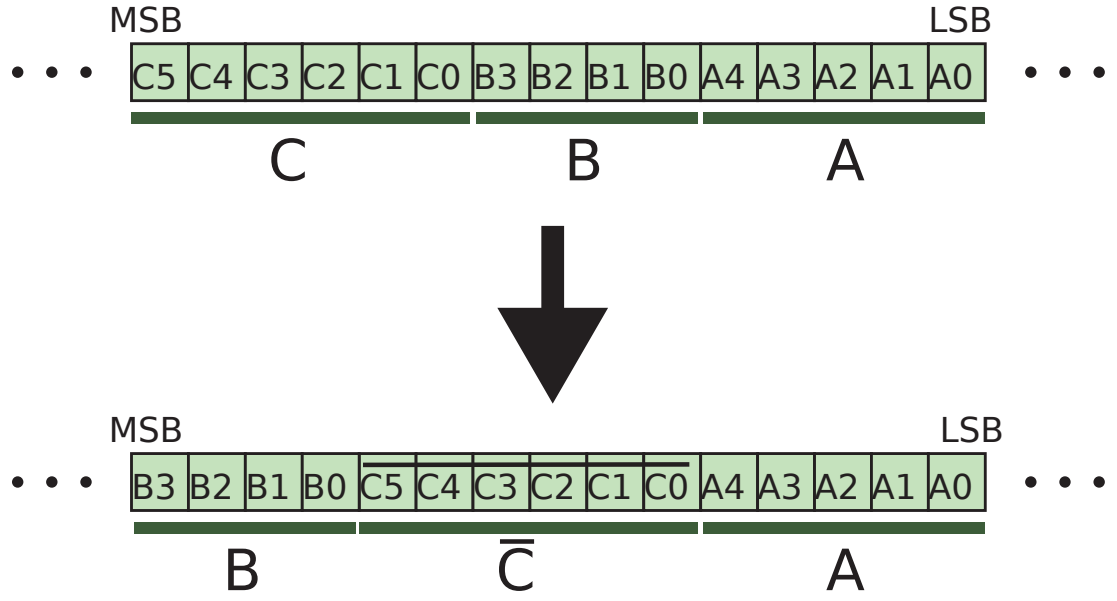

Figure 6: *Manipulation of data prior sort, where MSB and LSB denote most and least significant bits, respectively. In this case, data will be sorted in the order of B, C (in reverse), and then A. Note that* $(\overline{C} \equiv 2^6 - C)$ *is the two's complement of C. Upon completion of the sort, data is restored to its original form.*

3.5 Geographical Extensions

A frequent task in MDA involves categorizing a set of geographic locations, such as those attributed to contacts, based on whether they fall within a predefined area of interest. In this context, geographical regions may be specified in terms of a circumscribing path defined by a series of waypoints (latitude-longitude pairs). For small regions on the earth's surface, this problem may be approximated by way of a bounding polygon defined in a two-dimensional Cartesian coordinate system. However, for larger regions and/or those near the Earth's poles, this approximation can become grossly inaccurate. In this case, it is preferred to use *spherical polygons* defined on the surface of a sphere (the Earth). The curvilinear segments connecting way points are thus segments of great-circle arcs (geodesics). While contributing to the algorithmic complexity, this approach ensures accuracy for problems defined by any spherical polygon at minimal cost in execution time. As in the case with their two-dimensional counterparts, spherical polygons may be decomposed into a set of non-overlapping triangles. Determining whether a geographic location falls within an n-point spherical polygon thus simplifies to testing for inclusion in n spherical triangles. The steps involved in triangle decomposition and inclusion testing are briefly described here.

In what follows, analysis can be simplified by expressing a given geographical coordinate

as the unit vector

$$\begin{aligned} \hat{a}_x &= \cos\theta \sin\phi \\ \hat{a}_y &= \cos\theta \cos\phi \\ \hat{a}_z &= \sin\theta \end{aligned} \tag{1}$$

where ϕ and θ are the point's latitude and longitude, respectively (Fig. 7). In turn, two such vectors may be used to define a geodesic arc (Fig. 8), where $\alpha < \pi$ is assumed hereafter. A pair of arcs given by the unit vector pairs $(\hat{a}_1, \hat{a}_2)$ and $(\hat{b}_1, \hat{b}_2)$ may be tested for intersection by solving the linear system

$$\beta\hat{a}_1 + \gamma\hat{a}_2 + \delta\hat{b}_1 + \varepsilon\hat{b}_2 \tag{2}$$

for β, γ, δ, and ε. As this system is underdetermined, ε may be set to one, and the remaining unknowns may be recovered as

$$\begin{bmatrix} \beta \\ \gamma \\ \delta \end{bmatrix} = \begin{bmatrix} \hat{a}_1 & \hat{a}_2 & \hat{b}_1 \end{bmatrix}^{-1} \hat{b}_2 \quad . \tag{3}$$

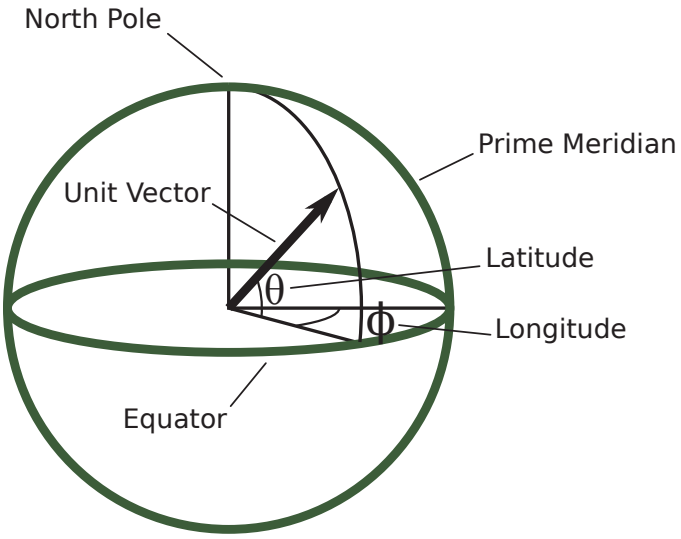

Figure 7: *Definition of geographical unit vector.*

When the solution of (3) lies in octant I of $\mathbb{R}^3$ (all entries are positive), the two great-circle arcs intersect at the point given by

$$\hat{a}_{\text{intersection}} = \delta\hat{b}_1 + \hat{b}_2 \tag{4}$$

The following theorem states the conditions under which a spherical polygon allows a decomposition into spherical triangles.

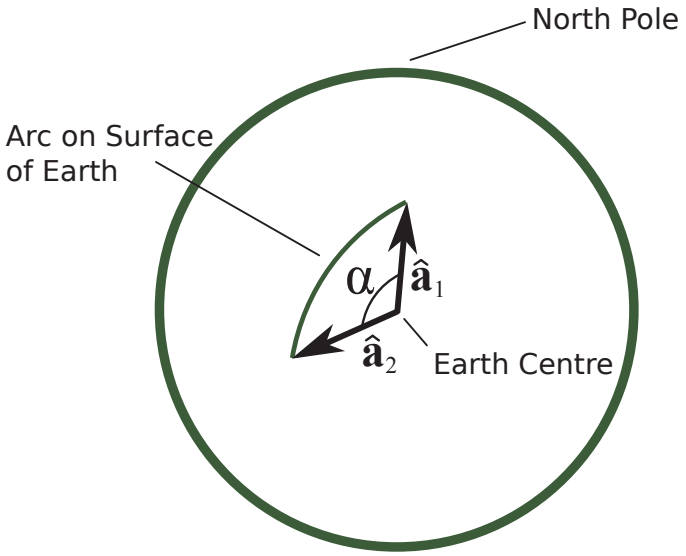

Figure 8: *An arc on the surface of the earth defined by two unit vectors,* $\hat{\mathbf{a}}_1$ *and* $\hat{\mathbf{a}}_2$.

Theorem 3.1. *Let P be a spherical polygon on a unit sphere S each of whose edges have length strictly less than* π*. If a side of P does not contain a great circle, then that side has a spherical triangulation [6].*

Proof. See [7]. □

The foregoing theorem ensures that a triangulation may always be found but does not prescribe the means for doing so. Nonetheless a procedure (Algorithm 2) may be derived by way of 'ear clipping', a process that iteratively cleaves a single triangle from the polygon until triangulation is complete.

Each member of the resulting set of spherical triangles is defined by its vertices, which may be expressed as the triplet of unit vectors $(\hat{\mathbf{t}}_1, \hat{\mathbf{t}}_2, \hat{\mathbf{t}}_3)$. In a manner similar to testing for the intersection of two arcs, inclusion of a geographic coordinate $(\hat{\mathbf{p}})$ in a spherical triangle T may be tested by solving

$$\mathbf{v} = \begin{bmatrix} \hat{\mathbf{t}}_1 & \hat{\mathbf{t}}_2 & \hat{\mathbf{t}}_3 \end{bmatrix}^{-1} \hat{\mathbf{p}} \quad . \tag{5}$$

Elementary geometry may be used to show that $\hat{\mathbf{p}}$ falls within T when the solution lies in the first octant of $\mathbb{R}^3$.

The software's geographical extension allows coordinates, arcs, spherical triangles, and spherical polygons to be defined, and arbitrary coordinates to be tested for inclusion in polygon-defined regions. As the matrix inversion is the most computationally expensive calculation in solving (5), this step is performed at the time of triangle definition, and the result is stored within the triangle object.

```
Data: WaypointArray
Result: SphericalTriangleArray
Initialization;
/* Step 1:  Generate unit vectors from lat/long          */
for i ← 1 to length(WaypointArray) do
  | UnitVectorArray[i] ← ComputeUnitVector(WaypointArray[i]);
end
/* Step 2:  Construct geodesics from unit vectors        */
for i ← 1 to length(UnitVectorArray) do
  | len ← length(UnitVectorArray);
  | GeodesicArray[i] ←
  |   ConstructGeodesic(UnitVectorArray[i], UnitVectorArray[(i+1) mod len]);
end
/* Step 3:  Ensure no intersection of geodesics          */
/* (ensure a simple spherical polygon)                   */
for i ← 1 to length(GeodesicArray) do
  | for j ← 1 to length(GeodesicArray) do
  |   | if i ≠ j and Intersect(GeodesicArray[i], GeodesicArray[j]) then
  |   |   | TerminateWithError;
  |   | end
  | end
end
```

Algorithm 2: Triangulation of spherical polygon (part 1 of 2)

```
/* Step 4:  Triangulate polygon defined by waypoints    */
tmpUVArray[] ← UnitVectorArray[ ];
while length (tmpUVArray) > 3 do
    for i ← 1 to length(tmpUVArray) do
        len ← length(tmpUVArray);
        tmp ← ConstructGeodesic(tmpUVArray[i], tmpUVArray[(i+2) mod len]);
        found ← false;
        for j ← 1 to length (GeodesicArray) do
            if Intersect (tmp, GeodesicArray[j]) then
                found ← true;
                break;
            end
        end
        if not found then
            first ← tmpUVArray[i];
            second ← tmpUVArray[(i+1) mod len];
            third ← tmpUVArray[(i+2) mod len];
            tmpTriangle ← ConstructTriangle (first, second, third);
            SphericalTriangleArray[ ].AddNew (tmpTriangle);
            tmpUVArray.Delete (second);
            break;
        end
    end
end
```

Algorithm 2: Triangulation of spherical polygon (part 2 of 2)

4 Application of Software to Data from Global Position Warehouse

A preliminary evaluation of the algorithm was undertaken by analyzing 200 million contact records from the unclassified Global Positional Warehouse (obtained from N6, MARLANT, and spanning approximately 5 years) using a Dell Precision M4600 laptop computer. Records were decimated so as to contain only contact position, time and date, thereby reducing memory requirements to 2.8 GB. A dual-threaded client program was written to classify contacts based on the ocean of origin (Atlantic, Pacific, Arctic, Southern, Indian, or none). The program employed the geographical extensions derived in §3.5 using hand-drawn spherical polygons to represent each ocean (Table 2).

The results of this analysis are shown in Fig. 9. Only half of the records originate in one of the five polygons, likely because the conventional definitions of the world's oceans (used to prepare Table 2) exclude many coastal regions of high vessel traffic and mooring, particularly in Asia and Europe. Of striking curiosity is the seemingly large proportion of contacts originating from the Arctic and Southern oceans. One conceivable explanation (to be confirmed with follow-on analysis) draws on the nature of satellite AIS reception, which is broken by periods of non-access when transmitted messages are lost. These periods are at a minimum at the poles and increase as latitudes approach the equator, a phenomenon that results from the polar orbits of AIS satellites. In view of the fact that AIS reports constitute the overwhelming majority of contacts in GPW, and because satellite reception serves as the primary means by which AIS messages are received over open ocean, it is plausible that the large number of contacts at high and low latitudes reflect the comparatively low message-loss rate in these regions.

The complete analysis (that involved 200 million decompressions, and one billion polygon-inclusion tests) required approximately one minute and forty seconds to complete, yielding a processing speed of $\sim$1 million records per second. These results suggest that analysis on tens of billions of records is very feasible on computers equipped with adequate memory.

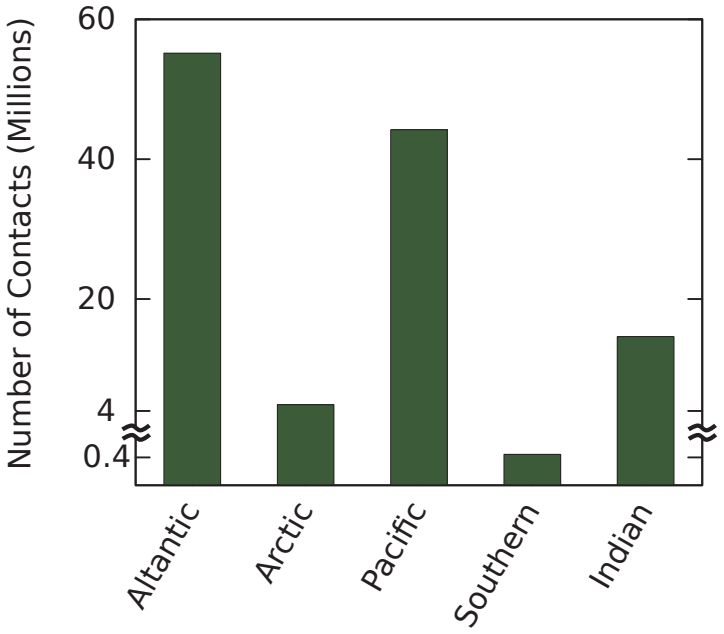

Figure 9: *Contacts originating from each of the world's oceans. The number of contacts originating from the Atlantic, Arctic, Pacific, Southern, and Indian oceans are (in millions) 55.2, 4.9, 44.2, 0.4, and 14.6, respectively.*

Table 2: *Waypoints (in degrees latitude and longitude) defining ocean boundaries. As there exists no consensus on the extent of the Arctic ocean, a simplified definition (similar to that of the Southern ocean) was used.*

Waypoint #	Atlantic	Pacific	Arctic	Southern	Indian
1	(-60.0, 25.9)	(-59.5, -72.4)	(65.0, 0.0)	(-60.0, 0.0)	(-60.0, 146.5)
2	(-33.8, 24.1)	(-25.2, -65.6)	(65.0, 20.0)	(-60.0, 20.0)	(-43.4, 146.4)
3	(-13.9, 15.0)	(1.4, -73.1)	(65.0, 40.0)	(-60.0, 40.0)	(-34.6, 116.0)
4	(5.8, -11.2)	(9.1, -78.6)	(65.0, 60.0)	(-60.0, 60.0)	(-24.9, 115.1)
5	(22.8, -13.6)	(8.2, -81.1)	(65.0, 80.0)	(-60.0, 80.0)	(-16.5, 126.7)
6	(33.7, -4.2)	(14.4, -90.4)	(65.0, 100.0)	(-60.0, 100.0)	(-10.1, 120.7)
7	(48.1, 0.4)	(29.9, -115.1)	(65.0, 120.0)	(-60.0, 120.0)	(-6.1, 103.4)
8	(53.1, -9.3)	(47.0, -123.0)	(65.0, 140.0)	(-60.0, 140.0)	(4.8, 94.7)
9	(63.3, -13.8)	(59.9, -138.2)	(65.0, 160.0)	(-60.0, 160.0)	(7.1, 82.8)
10	(64.4, -22.3)	(54.2, -164.7)	(65.0, 180.0)	(-60.0, 180.0)	(-0.8, 72.9)
11	(64.1, -40.7)	(51.9, 178.9)	(65.0, 200.0)	(-60.0, 200.0)	(11.1, 51.2)
12	(61.2, -44.0)	(56.2, 164.0)	(65.0, 220.0)	(-60.0, 220.0)	(-5.0, 38.1)
13	(60.1, -64.7)	(44.1, 149.4)	(65.0, 240.0)	(-60.0, 240.0)	(-33.8, 24.1)
14	(51.6, -63.8)	(30.4, 132.2)	(65.0, 260.0)	(-60.0, 260.0)	(-60.0, 25.9)
15	(48.9, -55.0)	(21.7, 122.2)	(65.0, 280.0)	(-60.0, 280.0)	(-60.0, 45.9)
16	(44.6, -69.6)	(0.8, 132.3)	(65.0, 300.0)	(-60.0, 300.0)	(-60.0, 65.9)
17	(32.9, -82.9)	(-12.5, 152.2)	(65.0, 320.0)	(-60.0, 320.0)	(-60.0, 85.9)
18	(27.3, -81.4)	(-49.1, 145.0)	(65.0, 340.0)	(-60.0, 340.0)	(-60.0, 105.9)
19	(24.6, -72.1)	(-60.5, 147.1)	–	–	(-60.0, 125.9)
20	(17.8, -61.4)	–	–	–	–
21	(8.8, -60.6)	–	–	–	–
22	(-9.7, -40.7)	–	–	–	–
23	(-30.2, -53.6)	–	–	–	–
24	(-60.0, -72.4)	–	–	–	–
25	(-60.0, -50.0)	–	–	–	–
26	(-60.0, -30.0)	–	–	–	–
27	(-60.0, -10.0)	–	–	–	–
28	(-60.0, 10.0)	–	–	–	–

5 Conclusion

The present work developed an in-memory database system that allows efficient statistical calculations on extremely large sets of maritime data. Processing large data sets typically entails querying relational databases, often requiring users to carefully design analysis and/or apply advanced programming practices to avoid the significant latencies that result from non-sequential disk access. Through bitmap data storage, dictionary compression, and further measures that reduce redundancy in maritime data, the software enables maximum use of system memory, thereby allowing very large data sets to be processed without hard-disk access. The capabilities of the system, including geospatial extensions, were demonstrated through a user-defined program that rapidly computes the ocean of origin for 200 million contacts in the MARLANT N6 Global Positional Warehouse. The simple application programming interface will enable users to rapidly develop and execute a variety of statistical/data analyses. The software is expected to facilitate deeper analysis in maritime domain awareness research and development, operational research and analysis, and real-time calculations in operational settings.

This page intentionally left blank.

References

[1] Y. Gauthier, S. Bourdon, LCdr S. Doré, and V. Fong (2004), Defining and Selecting Metrics for Intelligence, Surveillance, and Reconnaissance (ISR), DOR(MLA) Research Note RN 2004/08, Department of National Defence Canada, Operational Research Division, Directorate of Operational Research (Maritime, Land & Air).

[2] J. Bhattacharya (2010), Rudiments of Computer Science, Academic Publishers.

[3] J. Ziv and A. Lempel (1978), Compression of Individual Sequences via Variable-Rate Coding, *IEEE Transactions on Information Theory*, 24(5), 530–536.

[4] N. Josuttis (2012), The C++ Standard Library: A Tutorial and Reference, 2 ed, Upper Saddle River, New Jersey: Addison-Wesley Professional.

[5] B. Schäling (2011), The Boost C++ Libraries, Laguna Hills, California: XML Press.

[6] J. O'Rourke (2008), Computational Geometry Column 51, *ACM SIGACT News*, 39(3), 58–62.

[7] U. Brehm and W. Kühnel (1982), Smooth Approximation of Polyhedral Surfaces Regarding Curvatures, *Geometriae Dedicata*, 12(4), 435–461.

This page intentionally left blank.

DOCUMENT CONTROL DATA

(Security markings for the title, abstract and indexing annotation must be entered when the document is Classified or Designated.)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Defence Research and Development Canada – Atlantic
PO Box 1012, Dartmouth NS B2Y 3Z7, Canada

2a. SECURITY MARKING (Overall security marking of the document, including supplemental markings if applicable.)

UNCLASSIFIED

2b. CONTROLLED GOODS

(NON-CONTROLLED GOODS)
DMC A
REVIEW: GCEC APRIL 2011

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.)

In-Memory Analysis of Maritime Data Sets

4. AUTHORS (Last name, followed by initials – ranks, titles, etc. not to be used.)

Schaub, D. E.

5. DATE OF PUBLICATION (Month and year of publication of document.)	6a. NO. OF PAGES (Total containing information. Include Annexes, Appendices, etc.)	6b. NO. OF REFS (Total cited in document.)
April 2014	36	7

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical Memorandum

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

Defence Research and Development Canada – Atlantic
PO Box 1012, Dartmouth NS B2Y 3Z7, Canada

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)	9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.)
06eo	

10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.)	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)
DRDC Atlantic TM 2013-211	

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

(X) Unlimited distribution
( ) Defence departments and defence contractors; further distribution only as approved
( ) Defence departments and Canadian defence contractors; further distribution only as approved
( ) Government departments and agencies; further distribution only as approved
( ) Defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11)) is possible, a wider announcement audience may be selected.)

Unlimited

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

The present work develops an in-memory database system that enables efficient statistical calculations on extremely large sets of maritime data. The conventional approach of interspersing calculations with queries to relational databases incurs significant latencies when data retrieval patterns induce inefficient hard-disk operations. Mitigation through re-ordering data access is often feasible but at the expense of significant additional development involving practices in advanced computer programming. As random access of computer memory is several orders of magnitude faster than secondary storage access, it is highly desirable—when possible—to store databases exclusively in memory. Through bit-shifting, dynamic dictionary compression, and domain-specific data reduction techniques, the present work demonstrates a functional data storage and processing system capable of storing the entire Global Positional Warehouse in the memory of a personal computer. Data are made available to the user through a simple application programming interface that allows rapid development of efficient, large-scale statistical analysis programs.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Maritime Situational Awareness (MSA), Maritime Domain Awareness (MDA), Data Analysis, Database, Operations Centre, Operations Research (OR), Anomaly Detection, Data Mining, Pattern of Life

This page intentionally left blank.

Defence R&D Canada

Canada's leader in defence
and National Security
Science and Technology

R & D pour la défense Canada

Chef de file au Canada en matière
de science et de technologie pour
la défense et la sécurité nationale

DEFENCE DÉFENSE

www.drdc-rddc.gc.ca