Design Issues in Nearly Real-Time Meteorological-Data Systems at Local Sites

GEORGE J. HUFFMAN AND JAMES N. COOPER

Department of Meteorology, University of Maryland, College Park, Maryland

30 October 1987 and 11 July 1988

ABSTRACT

A scheme developed at the University of Maryland is described to illustrate the high-level design decisions which must be made in order to fully utilize the asynchronous character-oriented data transmitted over the Domestic Data Service. We list the major functions which nearly real-time meteorological-data systems must perform at local sites, and then emphasize solutions to the high-level design problems implicit in these functions, including processing sequence, data quality control, on-line data file management, off-line data archive content, and archive format.

1. Introduction

Many meteorological facilities around the United States have recently started to introduce computer-based nearly real-time meteorological-data systems. These systems promise a new level of flexibility and sophistication in using data for meteorological training, research, and operations. However, they frequently require many design decisions to be made at the local site, depending on local resources and requirements.

One impetus for the creation of computer-based systems has been the reorganization of the NWS's external data dissemination into the Family of Services (FOS), and the introduction of the FAA 604 data line. The FOS and FAA 604 provide data products at moderate speed, but without a strict schedule, so some computer-based processing is necessary to prevent massive printing or filing of unwanted information. Such processing has been extended to sophisticated systems that "decode" Surface Airway Observations (SAO) and rawinsonde reports into computer-compatible formats (integers, reals, etc.), and use them for digital processing and graphical display. On the other hand, relatively little attention has been paid to the steps which are necessary to capture the "raw" (or original) data and present them in various forms. In this paper we illustrate the high-level design issues which must be addressed in building a raw-data system by describing the raw-data scheme developed for the Meteorological Analysis, Processing, and Remote Communication (MAPR) System (Huffman 1988).

2. Background

During 1983 several faculty and staff in the Department of Meteorology, University of Maryland started designing a real-time data system which would provide the known benefits of digital analysis software on the emerging technology of scientific workstations. The GEMPAK system (desJardins and Petersen 1985) proved to be a good choice for handling the decoded data, but the raw-data system had to be designed from scratch for our choice of hardware. Three early choices govern the character of our system. First, we chose the FOS Domestic Data Service (DDS) to provide alphanumeric data. It carries data, discussions, and forecast material for North and Central America (WMO Regions VI and V) and surrounding waters (Friday 1983) in an asynchronous, character-oriented stream of data at 1800 b/s. The information is organized into "bulletins," short (usually less than a page) reports or groups of reports of a single data type. Bulletin contents are identified by a "header line," providing a WMO bulletin identifier, the station of origin, and a timestamp giving the UTC day/hour/minute of issue. No error detection is provided beyond parity bits.

Our second early choice was to target networked scientific workstations as the MAPR System's hardware environment. Although such workstations provide a rich environment, our raw-data system only depends on a few of the workstations' attributes. These are:

1) true multitasking, to allow simultaneous data reception and processing;
2) variable process priorities, ensuring timely attention to real-time data capture;
3) an archival device, in our case a ½" magnetic tape drive;
4) sufficient computing power to perform all the functions in a timely manner;
5) a tree file structure, although slight design changes could eliminate this requirement.

These attributes may be found on most mainframes or networks of scientific workstations, so we believe

Corresponding author address: Dr. George J. Huffman, NASA/ GSFC Code 612, Greenbelt, MD 20771.

© 1989 American Meteorological Society
that the discussion below has the potential for wide application.

Our final early choice was the functional organization of the system. Sets of applications (such as GEMPAK) are well suited for extracting information from structured data files, so the raw-data system has to do "everything else." That is, it must catch the data accurately, pass it to various destinations (filing routines, printers, or decoders), and manage the on- and offline data storage. Implementing these raw-data system functions requires a wide range of design choices. The bulk of this paper will focus on several high-level design choices which most designers face; processing sequence, data quality control, on-line data file management, off-line data archive content, and archive format. In each case, the solution developed for the MAPR System is probably not optimal at all sites, but our discussion should serve to expose the crucial elements of the issue.

3. The raw-data processing sequence

The task of capturing and routing the raw data is sufficiently complex that it is generally broken into a number of steps, as illustrated in Fig. 1 for the MAPR System. The data stream is passed through a locally-designed "speed converter" (which boosts the bit rate from 1800 to 2400 b/s) and fed directly into a standard serial input/output (I/O) port on a fileserver (a microcomputer dedicated to file access and communications). Within the fileserver the work is distributed over several "processes," which are independent, simultaneously-active programs.

First, the "Ingestor" process monitors the I/O port and writes characters to temporary storage (shared memory). The Ingestor process requires a high priority compared to other processes because it has to catch incoming data in real time, before the I/O port hardware buffer overflows. The "Digestor" process picks up completed bulletins and sends them to various "Disposer" processes according to a user-defined table of WMO headers. Options include queueing for immediate printing, decoding, or filing as text. The immediate printing option is self-explanatory, while a discussion of decoding is beyond the scope of this paper; we next consider some design issues arising in text filing.

The prime issue in designing a text filing system is the file structure, since that governs the amount of processing needed to store and retrieve data. For example, placing all of the data in a single file is easy, but searching for particular bulletins in that file is tedious. The alternative is to maintain multiple files keyed to bulletin timestamp and/or WMO header. In the MAPR System the files are arranged in a tree file structure (Fig. 2), in which the top-level directories are data type groupings (WMO headers, shown in Table 1), the second-level directories are timestamp (UTC) date, and the files themselves are specified by timestamp (UTC) hour. The composition and choice of data types might be different at other sites, but the idea is to group similar kinds of data to limit the number of files. The type-date-hour hierarchy is optimal for accessing several hours of the same kind of data on the same day.

One penalty of adopting multiple sequential files (compared to using circular files) is that the text-filing process must dynamically build and delete directories and files to avoid unbounded growth in the data storage requirements. When a bulletin arrives whose WMO identifier appears in Table 1, its type-date-hour combination is compared to the existing file system. If a match is found, the bulletin is appended to the appropriate file. Otherwise, an appropriate directory-file combination is created. Whenever a new date directory is created, the oldest date is deleted in the type directory to keep the number of dates below a user-defined limit (currently eight).

In use, an important benefit of the multiple sequential file structure becomes apparent. The text files may

---

1 The terms "Ingestor," "Digestor," and "Disposer" are used as defined in the Unidata Project (e.g., Campbell 1987).
be copied, read, and/or written (given the appropriate access rights) with standard operating system services since they have the ordinary sequential ASCII format. A simple copy-and-read "browse" program is the only application which we have found necessary, to prevent read/write conflicts between users and the text-filing process.

Eventually, old data must be deleted to provide room for new data, so future requests must be anticipated by generating a selected archive before the files are removed. The MAPR System archival activity (described in section 5) is executed interactively to help ensure quality control on the resulting archives. For the most part, implementing the processing-sequence components we have just described required straightforward, if somewhat time-consuming system and application programming. However, quality control on the header timestamps, and the archiving scheme presented high-level design problems which are endemic to real-time receive-only data systems. Accordingly, we discuss these two topics in more detail in the next two Sections.

4. Quality-controlling bulletin timestamps

For the most part errors in the DDS data stream pass unnoticed through the Ingestor and Digestor processes (because only bulletin headers are actually processed), and it is considered the Disposer processes' task to handle errors they might encounter. However, header errors do need to be resolved in the Digestor. In principle, the text of the bulletin might be scanned to provide data type and timestamp information for checking and correcting the header. However, we have not implemented such a scheme because the number of bulletin formats is so large. Accordingly, the header is taken "as is." Errors in the WMO identifier almost always cause the bulletin to be rejected as "unknown" because only a few user-specified alphanumeric strings (the WMO identifiers) are recognized.

The header timestamp presents a different problem. Many erroneous timestamps show obvious damage (i.e., contain something other than exactly six numeric characters). These cases are rejected. However, in other cases the header timestamp can be damaged but still look valid. Without further checks, experience shows that noise on the data line will quickly wipe out directories of good, sequenced data in favor of sparse, randomly-named files. The solution is to check the timestamp against a reference time, but choosing the reference time turns out to be difficult. The obvious choices of either the timestamp corresponding to the current system clock value (System Clock Timestamp, referred to as S) or the last valid bulletin header timestamp have their own problems:

1) Valid timestamps from consecutive bulletins on the DDS can vary from each other by 18 h or more.
2) Data transmission might have been delayed, causing the current bulletin's timestamp to be behind S (nonetheless, the data are queued up and will eventually appear).
3) Our raw-data system or some part of the data delivery system might have been inoperative, causing a loss of data and a gap in time between the last valid timestamp received and the current (valid) timestamp.
4) The system clock might be incorrect (a condition not currently tested).

The apparent errors just listed are generally associated with good sequenced data, so a scheme for catching truly corrupted timestamps needs to accept a variety of apparently anomalous header timestamps.

The algorithm we have developed for quality controlling timestamps recognizes both reference times mentioned above, the System Clock Timestamp, S, and the "Context Timestamp" (usually the last valid header time, referred to as C). In addition, tolerances are built around and between the references (see Fig. 3).

During steady transmission, a good timestamp should be "close" to C (i.e., fall between $C_t$ and $C_b$), in which case C is reset to the new timestamp. Here $C_t$ has been set to -21 hours to account for ship reports, which can be transmitted as much as 18 hours after observation time. The $C_b$ has been set to +1 hour to catch bulletins sent with the upcoming hour in the
timestamp. Thus, when data transmission is delayed, the system will correctly admit good, but delayed data no matter how long the wait.

On the other hand, when data have been lost, due to line or equipment failure, \( C \) might be much out of date by the time service is restored. In that case, \( S \) is the correct reference, with a forward limit \( S_a \). The \( S_a \) was chosen to be +1 hour for the same reasons as \( C_a \). When the timestamp falls between \( S \) and \( S_a \), \( C \) is reset to the current bulletin’s timestamp.

We choose not to define an \( S_b \) (in parallel with \( C_b \)), because there is some ambiguity about timestamps in the region between \( C_b \) and \( S \) (Region 3) when \( C_b < S \): A timestamp in this zone might be corrupted or it might indicate that \( C \) is in error. We choose to capture as much good data as possible, even at the risk of taking some faulty data, so we accept timestamps in Region 3. However, we require that three consecutive timestamps appear in Region 3 before \( C \) itself is reset to the current bulletin’s timestamp. This test reduces the chances that a corrupted timestamp will move \( C \) far ahead of the “true” context that good data are lost in Region 2. This problem is generally not a serious issue when \( C_b = -21 \) h, but would be for smaller values of \( C_b \).

In practice, the \( S \) test is performed first to preclude a pathological condition in which timestamps ever farther in the future could move \( C \) far ahead of \( S \). The \( S \) test summarized rejects timestamps ahead of \( S_a \) (Region 1). Then \( C \) is used to reject timestamps behind \( C_b \) (Region 2). Finally, all timestamps falling between \( C_b \) and \( S_a \) are accepted.

The \( C \) is maintained as an internal variable and written to a file on normal shutdown. On initialization, \( C \) is read, if found; otherwise, \( S \) is used. Currently \( S \) is not quality controlled, but if we did, it would involve a similar scheme for maintaining the previous call to the system clock as an internal variable, which would be stored in a file on shutdown and read back in on startup.

5. Archival system

The final disposition of the data occurs in the archiving routine. It is perhaps at this point that the most local tailoring needs to be done. The extreme cases of archiving, either to save nothing or to save “everything,” are both hard to justify at a university. The ongoing need for case studies and exercises mandates some local storage, while resource limitations usually prohibit a complete archive. Our compromise in the MAPR System is to routinely archive data in which we might expect future interest, together with the option for special cases which are flagged as they occur.

Even in real time we do not capture all available data. Synoptic reports (which are largely redundant with SAOs), terminal forecasts, Manually Digitized Radar, climatological reports, river stages and forecasts, and most worded forecasts are all absent from Table 1. Additionally, only a reduced set of SAOs and SXs is collected in raw form outside of 00 and 12Z (although our decoded SAO dataset carries all U.S. stations at every hour). The data captured in real time aggregates to about 2.3 Megabytes per day, some 40% of the total available on the DDS. A subjective survey of expected data requests led us to further omit all “text” bulletins from the archive, as well as all “surface” bulletins outside of 00, 06, 12 and 18Z. However, all “ship” and “upper air” bulletins are retained, since those data sets are intrinsically sparser, increasing the need for off-hour reports. The final archived selection of “forecast,” “ship,” “upper air” and “surface” aggregates to 1.4 Megabytes per day.

If we require that the archive be compact, durable, fault-tolerant, and easy to use, then half-inch magnetic tape is the medium of choice, at least until compact disks are widely available. Furthermore, if we restrict ourselves to writing fixed-record, fixed-block, ASCII records on unlabeled tapes, our experience shows that the tape can be read at most sites. The record length is chosen as 80 characters (bytes), since that is the maximum legal line size on DDS. The individual tape files are chosen to be images of the on-line data files. We avoid more compact schemes, such as blocking together all hours in a day (“compact files”), since these schemes would require more computations to retrieve data.

The last parameter needed to carry out tape archiving is block size. The following discussion develops a quantitative simulation for choosing a block size that minimizes the tape space required to store a given set of files. We close by applying the simulation to our particular archiving scheme for 43 complete days of data.

The length of tape, \( t_i \), needed to archive a particular file may be expressed as

\[
t_i = N \left( \frac{F_i}{B} \right) \cdot B \cdot \left( \frac{R}{D} \right) + \left( N \left( \frac{F_i}{B} \right) + n \right) \cdot G
\]  

(1)

\(^2\) Additionally, detailed forecasts and hourly data from Maryland and surrounding areas are being archived on hard copy to assess the usefulness of this format.
where $F_i$ is the file length (in records), $R$ is the record length (in bytes), $D$ is the tape density (in bytes per inch), $B$ is the block length (in records), $n$ is the number of interrecord gaps between files, $G$ is the interrecord gap size (in inches), and $N$ is the least-greater integer function, which rounds fractions up to the next whole number:

$$N[x] = \begin{cases} x, & \text{mod}(x, 1) = 0 \\ x + 1 - \text{mod}(x, 1), & \text{mod}(x, 1) > 0. \end{cases}$$

(2)

In (2) mod is the modulus function.

The first term in (1) represents the space occupied by the data, together with some space left over at the end of the last block. As shown in Fig. 4, for fixed values of the other parameters (see Table 2) the space occupied by the data blocks is a nonlinear function of block length. Each jump represents a change in the number of blocks needed to represent the file, and the values on either side of the jump are $(F_i + B - 1) \cdot R / D$ and $F_i \cdot R / D$. The second term represents both the space occupied by gaps after each block, which is inversely related to block length, and extra file mark space at the end of the file, a constant. We see in Fig. 4 that the two terms tend to oppose each other, resulting in some range of block lengths which will minimize the tape length needed to archive the file.

In the more general case, the length of tape, $T$, needed to archive a string of files is just the sum of the individual lengths, $t_i$:

$$T = \sum_i t_i.$$  

(3)

For any realistic mix of data files with different lengths, the nonlinearity in (1) makes it hard to estimate an optimal block size analytically. Instead, we must take a sample of the data to be archived and compute (3) over a range of $B$. The test dataset should be as large as possible, in order to reduce the effects of anomalous conditions on individual days. For example, significant weather increases the volume of SAOs, but communication problems might reduce ship reports on another day. Days on which there were large amounts of missing data (usually because the MAPR System was inoperable) were omitted from the sample.

Our data sample was drawn from 8 weeks of nearly real-time data collected from December 1986 to February 1987, of which 43 days were considered sufficiently complete for use. A histogram of file lengths is displayed in Fig. 5. The large number of small files represents ships, off-hour surface, and off-hour upper airs, while the peaks at roughly 3000, 4000, and 6000 records represent major-hour upper airs, major-hour surface, and forecasts, respectively. Figure 6 displays the tape space requirements as a function of block length (solid line). Compared to the one-file result in Fig. 4, Fig. 6 is much smoother, no doubt due to the variety of transition points for the various files. We can

---

**Table 2. Parameter values for the archiving tests.**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_i$</td>
<td>100 records (single-file test)</td>
</tr>
<tr>
<td>$R$</td>
<td>Various (43 day sample)</td>
</tr>
<tr>
<td>$D$</td>
<td>80 B</td>
</tr>
<tr>
<td>$B$</td>
<td>6250 B/in</td>
</tr>
<tr>
<td>$n$</td>
<td>Variable</td>
</tr>
<tr>
<td>$G$</td>
<td>2 in</td>
</tr>
</tbody>
</table>

---

![Fig. 4. Length occupied on a 1/8" magnetic tape by a single file, given the parameters in Table 2. The difference between the total length and the length occupied by the data blocks is the length occupied by the inter-record gaps and file marks.](image)

---

![Fig. 5. Histogram of archive file lengths for the 43 day sample collected from the Domestic Data Service.](image)
conclude that a block length of 150 to 200 records (lines) will minimize the amount of tape used.

For comparison, the space requirements of two other archiving formats are also plotted in Fig. 6. "All files" shows the space used to save all of the bulletins in Table 1, grouped in the five data types shown. The additional files are mostly small, so the optimal block size shifts down to 100 or 150 lines. "Compacted files" shows the result of appending our archived data into just four files per day, one for each data type. This representation has a broad minimum between 400 and 600 lines, and is about 20% more efficient than our scheme. Although we choose to keep our scheme at present because data retrieval is simpler, another site might choose the compacted file scheme.

6. Summary

We have described the raw-data scheme which has been implemented on the MAPR System, illustrating the decisions that must be made in order to capture, route, store, and archive the raw data from the DDS.

The Ingestor process catches the data as they arrive in real time, and passes them to the Digestor process, which sends the data (a bulletin at a time) to various Disposers (decoding, printing, and raw-text filing). At this point the bulletin timestamp is checked for validity. The raw-text-filing scheme adopted in the MAPR System is a tree file structure based on data type, UTC date, and UTC hour, providing sequential ASCII files which can be manipulated with standard system utilities. The final stage in the data stream is archiving. We have presented a scheme for saving selected files on one-half inch magnetic tape, together with some test demonstrating the characteristics of several variants.

It is likely that the details of our system will change in the face of new requirements and opportunities, especially the delivery of the Unidata System for Scientific Data Management. Nonetheless, the same issues will continue to require answers, and the information developed here might be applicable to other data problems.

Acknowledgments. The Unidata Project and its Working Groups, particularly the Local Hardware-Software System Working Group, has provided an invaluable forum for collecting and refining the concepts in the MAPR System. Dr. David R. Rodenhuis and Mary desJardins have provided significant comment on the system, while Tony Gehr assisted in the original development work. Of course, the authors bear the responsibility for the MAPR System’s current form.

REFERENCES


