

Russell K. Rew\* and Anne Wilson  
 Unidata Program Center  
 University Corporation for Atmospheric Research  
 Boulder, Colorado

## 1. INTRODUCTION

Since 1995, with the assistance of a cooperative community effort, Unidata has been propagating near-real-time meteorological data to participants via its Internet Data Distribution System (IDD). The IDD delivers data continuously from multiple sources to educational and other institutions by using Unidata's Local Data Management (LDM) software. The LDM system is a collection of freely-available protocols and programs that provide for reliable event-driven data distribution as well as facilities to select, capture, process, and redistribute the data.

Recently the Unidata LDM has also been incorporated into several data distribution networks independent of the IDD, such as CRAFT (Droegemeier, 1999), SuomiNet (Ware, 2000), and GODAE (Dimitriou, 2001), and its use is being evaluated for other projects.

We summarize the current status of the LDM and its use in the IDD, including recent improvements and their effect on the scalability, performance, and reliability of the system. We also discuss remaining LDM limitations and directions for future development to keep the data flowing for education and research.

## 2. DATA FEEDS AND DATA PRODUCTS

The LDM software provides event-driven delivery of real-time data flows that encompass various sources of observed and model output data. Each participating IDD site may subscribe to one or more data feeds, and may (subject to redistribution restrictions) use the LDM software to relay selected data feeds to other downstream sites.

For use in the IDD, data in each feed is packaged into a sequence of *data products*, the smallest unit of data dealt with by the LDM. A data product represents a single named data item such as the report of an observation, a gridded data field, or an image. Data products can be as small as a few dozen bytes for a short bulletin or as large as many megabytes for a satellite image.

In addition to the data bytes, each data product

. \*Corresponding author address: Russ Rew, UCAR Unidata, P.O. Box 3000, Boulder, CO 80307-3000; e-mail <russ@unidata.ucar.edu>. The Unidata Program Center is sponsored by the National Science Foundation and managed by the University Corporation for Atmospheric Research.

includes some additional information:

- The *product identifier*, used in pattern matching rules to determine what to do with each product (filing, decoding, etc.),
- The *feed type* that indicates the naming convention and any distribution restrictions,
- The *signature*, a 16-byte MD5 checksum used to eliminate duplicate products, and
- A *time stamp*, specifying when the product was first injected into the IDD.

(Yoksas, 2001) describes some of the currently available IDD data feeds, including the volume of data and number of products available in each feed. As an example, one existing LDM system (motherlode.ucar.edu) ingests 15 IDD data feeds, including several experimental feeds not yet generally available, comprising an average hourly volume of about 1 Gbyte in about 80,000 products. This LDM host relays over 5.7 million data products per day to 17 other LDM systems.

The previous version of the LDM software could not have handled this volume of data. Below we describe scalability problems with LDM version 5.0 and a little of the computer science that went into the design of the new version in order to handle significantly higher data rates.

## 3. THE LDM PRODUCT QUEUE

Each LDM system uses a local data store called the *product queue*, which is shared among the LDM programs that get data from upstream hosts, read and process the data, and send data on to downstream hosts. Old products are expired out of the product queue when they are no longer needed. Product queues typically hold about an hour's worth of data from each data stream. This buffer of data provides the elasticity needed to support reliable data delivery in the face of outages and congestion. The LDM is designed so that a relay site can be shut down and restarted, then catch up from where it left off by getting the products it missed from the upstream host so that downstream sites will still receive all the subscribed products.

Despite its name, the product queue is more elaborate than a simple first-in-first-out queue of products. The LDM product queue must also support:

- Concurrent access by multiple processes;
- Detection of duplicates, so no product is inserted more than once;

- Ability to access or delete products that match feed type, product identifier pattern, and time stamp constraints;
- Ability to manage free regions, so space in the product queue is recovered and used efficiently; and
- Ability to quickly access data by order of insertion.

These requirements are met by a library that implements a product queue using a memory-mapped file open for shared access. The use of a memory-mapped file is important for performance of the system, but also ties the current implementation to Unix-based platforms. Inserting and deleting data products from the product queue requires maintaining indexes for access to products by insertion time, by signature, and by offset. Managing free regions in the queue requires maintaining additional data structures to find and coalesce free regions.

Inserting a new product into a full product queue includes the following steps:

- Determine whether the product is already in the queue by looking up its MD5 signature.
- Find the free region that is the best fit to the size needed to hold the product; if no such free region exists, delete oldest products not currently locked until a free region of adequate size is available, consolidating adjacent free regions when possible and updating the index to free regions by size.
- Split the best-fit free region so unneeded space goes back on to the free region list.
- Copy the product into the free region. Create an insertion time index entry and add it to the insertion time index. Create an MD5 signature index entry and add it to the signature index for future duplicate detection. Add the region to the index of in-use regions.

In LDM 5.0, the amount of time it took to insert or delete a product depended on the number of products in the product queue. Only the small fixed-size data structures that indexed the data products by time, MD5 signature, and offset were moved when a new product was inserted or an old product was deleted, and binary searches were used for fast look-up. When this software was first deployed, product queues typically held 10,000 or fewer products, so the time required for product insertion/deletion was insignificant, but with larger product queues the time required to move index elements to make space for an insertion or fill in the gap of a deleted entry became noticeable. As benchmarks below will show, LDM 5.0 can no longer keep up with the current rate of products when it has a queue that holds on the order of 100,000 products.

#### 4. IMPROVEMENTS FOR SCALABILITY

We developed LDM 5.1 by implementing the LDM 5.0 product queue library interfaces with new data struc-

tures and algorithms. Designing a data structure to support searching, insertion, and deletion efficiently is a classic computer science problem. Solutions include balanced-trees such as AVL trees, B-trees, and red-black trees. Such textbook solutions are not directly usable for LDM product queues, however, because the LDM data structures must be shared among multiple processes, and must be persistent so an LDM system can be stopped and restarted without losing data. This precludes the use of operating system dynamic storage allocation facilities, hence custom allocation and garbage collection algorithms for space in the product queue are required.

These additional complexities suggested the use of a simpler data structure, the *skip list* (Pugh, 1990), that achieves high performance by relying on “probabilistic balancing.” Pseudo-random numbers are used to keep the data structure balanced so that the times required for inserting, deleting, and searching by key increase as  $\log(N)$  instead of  $N$ , where  $N$  is the number of products in the product queue. For the new LDM, skip list algorithms and data structures were adapted for concurrent access to maintain the insertion time index and the free region size index. A variation of open hashing with chaining was implemented for maintaining the MD5 signatures of products in the queue for duplicate detection. We implemented data structures for dynamic storage management of regions within the queue by using a best-fit strategy supported by the index by extent to free regions (Wilson, 1995). Consolidation of adjacent free regions was required to prevent fragmentation, but the usual border tag algorithms for region consolidation were not available, because they would require extra locking and excessive paging. Instead, we used an auxiliary index of free regions by offset.

The overhead space needed to implement fast access to product signatures, to products by insertion time and offset, and to free regions by offset and extent was 68 bytes per product in LDM 5.1 compared to an overhead of 48 bytes per product in LDM 5.0.

#### 5. RESULTS

Table 1 compares LDM 5.0 with LDM 5.1 using measured steady-state rates for how many products per second can be inserted into a full product queue containing  $N$  products. These rates were observed in sending small products (single station METAR bulletins) from an LDM on one workstation host to an LDM on a second workstation host over a fast local area network (100 Mb/sec) with no other computing load on the destination system, so they represent a theoretical best-case that is unlikely to be achieved with actual data feeds that have larger products, higher network latency, and other sources of load such as decoding and relaying products.

In each case, the destination LDM is inserting new products into a full product queue, hence it is deleting oldest products as needed to make room for each new product. The important thing to notice is not just that the

insertion rate in LDM 5.1 is faster, but that it has almost no dependence on the number of products in the queue, so the new LDM is scalable to handle larger queue sizes.

**Table 1: Measured rate of small product transfer into a full product queue**

Products in Queue	Products/sec, LDM 5.0	Products/sec, LDM 5.1
20000	196	1073
40000	98	1087
80000	45	1083
160000	20	1062

From Table 1, it is apparent that an LDM 5.0 system with 80,000 products in its queue (about an hour's worth of data) would be able to keep up with ingesting small products at the current rate, but would lose data with a larger queue size or if other CPU-intensive tasks such as decoding data products were required. Although the current average rate from all 15 feeds is about 21 products per second, data products tend to arrive in bursts at rates as high as 200 products per second. Since LDM 5.0 could not handle products at this rate, it would necessarily add undesirable latency to the delivery of data to downstream sites whenever a large burst of products arrives or whenever arrival rate is higher than average for a sustained period.

A more realistic benchmark using real products for a typical hour from all current feeds is presented in Table 2, comparing average rates in products/second for sending the data (1.1.Gbytes in 81,000 products) from one host to another, using various product queue capacities at the destination host. The results again show a speedup of the new implementation that increases as there are more products in the destination queue.

**Table 2: Measured rate of typical product transfer into a full product queue**

Products in queue	Products/sec, LDM 5.0	Products/sec, LDM 5.1
20000	130	300
40000	66	243
80000	40	230
160000	20	210

Another feature available with the new version is the possibility of creating and using product queues larger than 2 Gbytes. On 64-bit platforms such as Sparc-v9,

Alphas, and IRIX64 systems, it is possible to build the LDM programs to support huge queues that use 64-bit offsets instead of the 32-bit offsets used in LDM 5.0, so that product queue size is limited only by the amount of available local disk space. Additionally, new feed types and synonyms for existing feed types were added to the LDM software to handle planned additions and changes to the IDD data streams.

## 6. REMAINING LDM LIMITATIONS

The most significant remaining limitations to the LDM software are lack of support for dynamic routing and a limited number of feed types. In addition, the use of unicast rather than multicast for data transport might be considered a limitation of the current architecture.

*Static routing:* Currently the static routing among IDD hosts for each feed type must be manually created and manually maintained. In addition, each host is assigned an upstream fail-over server that will provide data if the primary server crashes or becomes disconnected. For some important data streams such as NOAAPORT feeds, a ring of top-level source nodes each receive data via a satellite feed and also via another top-level node for redundancy. Geographical dispersion of top-level nodes provides reliability in the face of satellite solar interference periods. Nevertheless, the lack of dynamic routing support in the LDM protocols and architecture can potentially compromise reliability and usefulness of the system. Self-managing data flows that could adapt to provide near optimal routing would support some interesting new possibilities for the IDD.

*Non-extensible feed types:* The LDM supports subscription requests that specify an arbitrary subset of 32 predefined feed types. There is currently no facility for user-defined feed types or hierarchical feed types. As a result, subscriptions that require finer granularity must use pattern matching in the product identifiers to distinguish product feeds.

*Unicast transport:* When an LDM relay host sends a product to multiple downstream sites, the product is sent separately to each subscribing site. The product queue permits different sites to be fed at different rates; if a downstream site becomes disconnected and later reconnected to the network before any missed products are aged out of the upstream product queue, no products will be lost. Using multicast instead of unicast would provide a different solution for this problem, but achieving the same reliability for long outages with multicast may not yet be practical.

## 7. POSSIBLE FUTURE ARCHITECTURES

Among other alternatives, Unidata is considering Usenet software as a possible architecture for the next generation LDM. InterNetNews (INN) is a popular, open source implementation of the Network News Transport Protocol (NNTP) providing a suite of interacting programs for the distribution of news postings (ISC, 2000).

The similarities in functionality between INN and the LDM are numerous and striking. INN servers handle a comparable volume of data, since a full news feed is currently running just under 90 Gbytes/day and just over a million articles per day, with article latencies similar to LDM product latencies. Though originally designed for text, binary data now comprises about 94% of the daily traffic volume.

INN uses a data flooding approach, with deliberate redundant cross-connections to other hosts so that articles can reach hosts by multiple routes. This way, networks do not have to be precisely configured as the flooding algorithm ensures that an article eventually gets where it should go. With this approach, news finds the path of least resistance to get to its destination quickly. While routing is still static among INN hosts, there is no centrally maintained routing tree, just a routing network in which local changes can be made easily.

Two other attractive features of INN are the extensibility of hierarchically organized news groups to replace the limited number of LDM feed types and the possibility of making use of multicast infrastructure when and if multicast implementations of NNTP based news servers become widely available.

INN's biggest strengths are robustness and flexibility. The price of flexibility is complex configuration mechanisms. An additional drawback is the overhead of encoding required to transfer binary data in a system originally designed for text. We have developed and tested an encoding of IDD products that adds little overhead to large products and has cost comparable to the computation of the MD5 signature used for duplicate detection.

Further testing should determine whether an INN/NNTP approach will be practical. Whether it will be necessary may depend on whether another recent idea for improving the scalability of the IDD can be successfully implemented. This plan would simplify and reconfigure the IDD to make use of a number of well-configured and well-connected LDM servers at enough new top-level IDD relay sites to feed all other IDD sites so that additional lower-level relays would not be necessary (Yoksas, 2001).

## 8. CONCLUSION

The new LDM isn't just faster than the previous version by a constant factor, as could be achieved with conventional tuning and optimization techniques. Instead the use of superior algorithms actually improves the speedup as product queues get larger, thus enhancing the scalability of the system to handle more data.

These improvements to the LDM should provide enough excess capacity to give us the time to handle growth in the number of data feeds and data products for the near future, while we consider and compare alternative architectures that may eventually be needed.

## 9. REFERENCES

- Davis, G. P. and R. K. Rew, 1994. "The Unidata LDM: Programs and Protocols for Flexible Processing of Data Products". *Proceedings, 10th International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology*, Amer. Meteor. Soc., January, Nashville, Tennessee, pp. 131-136.
- Dimitriou, D., 2001. "The US GODAE Monterey Data Server." *Proceedings, 17th International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology*, Amer. Met. Soc., January, Albuquerque, New Mexico.
- Droegemeier, K.K., J. Zong, K. Brewster, T.D. Crum, H. Edmon, D. Fulker, L. Miller, R. Rew, and J. Martin, 1999: "The explicit numerical prediction of an intense hailstorm using WSR-88D observations: The need for realtime access to Level II data and plans for a prototype acquisition system". *15th International Conference on Interactive Information and Processing Systems (IIPS) for Meteorology, Oceanography, and Hydrology*, Amer. Meteor. Soc., January, Dallas, Texas, pp. 295-299.
- ISC, 2000. Web page: "INN: InterNetNews". Internet Software Consortium. <<http://www.isc.org/products/INN/>>
- Pugh W, 1990. "Skip Lists: A Probabilistic Alternative to Balanced Trees". *Communications of the ACM*, 33, pp. 668-676.
- Unidata's LDM Web page: <<http://www.unidata.ucar.edu/packages/ldm/>>.
- Ware, R. H., D. W. Fulker, S. A. Stein, D. N. Anderson, S. K. Avery, R. D. Clark, K. K. Droegemeier, J. P. Kuettner, J. B. Minster, and S. Sorooshian, 2000: "SuomiNet: A Real-Time National GPS Network for Atmospheric Research and Education". *Bull. Amer. Meteor. Soc.*, 81, pp. 677-694.
- Wilson, P. R., M. S. Johnstone, M. Neely, and D. Boles. 1995. "Dynamic storage allocation: A survey and critical review". *Proceedings of the International Workshop on Memory Management*. Kinross, Scotland, UK.
- Yoksas, T. and S. Worley, 2001. "UCAR's Realtime and Retrospective Data Access Project. *Proceedings, 17th International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology*, Amer. Met. Soc., January, Albuquerque, New Mexico.