Time Synchronization Over Networks Using Convex Closures

Jean-Marc Berthaud

***Abstract*—This paper presents a general time synchronization algorithm that analyzes the time offset between any two computers' clocks in a network and its evolution, by using mathematical topology properties. It builds a conversion function that produces precise and guaranteed bounds for each time conversion, and which provides accurate time synchronization. It is able to automatically adjust the observation process so as to maintain the error bound within some specified limit. It does not require adjustments to local clocks, and it features a way of filtering observation data based on a criterion of usefulness to improve precision, thus discarding only useless information.**

Advantages over approaches using other tools to filter out and analyze observation data (mean and variance, linear regression, midpoint functions, etc.) are exposed. Special attention is given to assessing the uncertainties and errors made in the observation process, and to their propagation in the estimation processes. The developed technique allows one to globally achieve a better precision than what has been reached on each single observation, given some conditions of operation that are explained.

***Index Terms*—Continuous estimation from discrete samplings, distributed processing, error propagation, network time synchronization.**

I. Introduction

CONSIDER a system of distributed processes over a network of communicating and independent devices, used, for example, to manage distributed databases, or handle coordinated actions distributed over several processors, or record events for time duration measurements, or distribute time of the day. They have one common feature: they all require some kind of time synchronization, to be able to refer to a common timescale when carrying out their work.

Definitions and Terminology on Time Synchronization: One device is said to be synchronized in time with another device if at any moment it can tell what is the time offset between its own clock and the other device's clock, or it can display the same time. The first device is called slave or client, and the second one is called master or reference. Time synchronization designates the process executed by one or more devices linked by a communication medium, so that some or all devices are synchronized in time with other devices through observation of their time.

In this paper, the terms "slave" and "master" will be used to refer to the roles defined in the above definitions. The term "host" will be used to refer to a device having a processor and a local clock, and running a time synchronization process. A host can play both slave and master roles in relation to different hosts at the same time.

Exact time synchronization over a network of computers cannot be achieved when there are uncertainties on transmission delays in the network, and on processing delays in the protocol layers of the computers [1], [6]. Unfortunately, this is the very vast majority of the cases, and thus uncertainties have to be managed to provide approximate time synchronization.

In addition, determining the offset to the master's time at one specific moment does not imply that a slave is synchronized with it after or before that moment. Indeed, independent computer clocks tend to evolve at different rates unless there is an atomic cesium clock in every host (this is referred to as clock drift).

Therefore, hosts attempting to time synchronize must constantly refresh their observation data and handle uncertainties to guarantee that they remain synchronized within a requested error interval.

A. Related Work

Techniques generally applied in networks regularly measure the offsets between clocks of the computers to synchronize. To avoid bursts of messages happening in narrow synchronization periods, some introduce randomness in the synchronization period [8], to stagger messages in time. Others [5], [7] let each synchronizing host decide its own synchronization rhythm.

Existing work can be divided into two major groups:

- In the first group, strong relationships are defined between hosts, specifying for each host some other hosts it can observe, and/or the ones it should not. In this hierarchical model, one or several special hosts are declared reference hosts, and provide the initial external time without looking at any other host. Examples of this kind of approach are encountered in the 4.3BSD *timed* protocol [2], the Internet Network Time Protocol (NTP) [5], [7], or the digital time service (DTS) [4]. These solutions have the advantage of reducing the number of synchronization messages over the network. They also allow the building of history of observations between slave and masters, to improve precision of time synchronization.
- In the second group, no static role is enforced for each host, but rather it is tried to get the set of hosts synchronized between them as a whole. Some special hosts can also be time synchronized with an external clock source [10]. This typically can lead to a quadratic number of messages being exchanged over the network. However, some

Manuscript received August 31, 1998; revised December 1, 1999; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor U. Shankar.

The author is with the IBM Networking Division, La Gaude Laboratory, 06610 La Gaude, France (e-mail: berthajm@fr.ibm.com).

Publisher Item Identifier S 1063-6692(00)03315-X.

work allows reducing this drawback [8]. The main advantage of such solutions is that they are more robust and support failures of a number of hosts in the network.

In both groups, some common principles are used.

Each time a slave (i.e., a host/process getting time information from other hosts/processes) acquire a new set of several observations, it tries to reduce this set by excluding what it thinks could be erroneous or less accurate observations. Observations are filtered out based on statistical considerations [4], [5], on assumptions regarding maximum number of failing hosts [9], or on some time-constraints to respect [8]. It can lead to getting more observations if the size of the reduced set is judged insufficient to produce a valid estimation.

Then, an estimation is produced from that set of observation(s), and several tools may be used in the estimation evaluation process, such as: mean value, weighted average, linear regression, midpoint functions, etc. (see [8] and [9] for an explanation of the last one). The assessment of the error made on that estimation, when there is one, is done accordingly: statistical value (variance, dispersion in [4] and [7]), maximum bound derived from constraint or sometimes from assumption [9], [10].

In [4] and [5], these two steps are improved using a history of selected observations.

At last, the estimation is used to determine the amount by which a slave should adjust its local clock. To reflect the adjustment, it can either modify its local clock, either produce a virtual clock that is calculated by adding that amount to the local clock. The new value output by the system is then considered valid until the next synchronization round. In both cases, the adjustment can be discrete [2], [4], [5], [8], or can be introduced continuously [3], [8].

B. *Paradigm Shift: Do Not Adjust Clocks, Measure Usefulness of Information, Provide Accurate and Continuous Error Bounding*

In the context of regular operations (no false-tickers), the main challenge that time synchronization techniques are facing is the calculation of uncertainties and the determination of optimal—or not far from optimal—and yet guaranteed error bounds when estimating a new time. That issue is not so strong when the application is just displaying the same time as the other computers. When this time has to be used for other more demanding fields such as distributed database merging, distributed processing or time elapsed between events, this becomes a major requirement.

When using statistics to filter out observation data or to produce an estimation, time synchronization processes are unable to supply an accurate estimation of the error made on time displayed by slaves. The resulting error interval is either statistically guaranteed, and hence they cannot guarantee that a requested error interval is satisfied, or it is large enough compared to results and uncertainties of the process, but in that case it is not optimal.

When assumptions or general constraints are used, the filter and bounds are calculated using these maximum values, or some value derived from them. Then, the length of the guaranteed error interval is also not optimal. Moreover, the errors actually measured are rarely incorporated.

In both cases, when a time synchronization process guarantees an error interval, its length is determined and fixed until the next synchronization point occurs. Since hosts' clocks are drifting from each other, the error interval could clearly be better tailored by being continuously function of time.

In any case, the process used to discard observation data can discard useful data. Since it is not able to measure the actual benefits an observation data is bringing to a time estimation, the criterion to filter it out is somewhat arbitrary. Indeed, even a seemingly erroneous data can contribute, when put together with other data and used correctly, to improve the overall result. In that sense, these processes are again suboptimal.

Last, adjusting the local clock (either for real or by using a virtual clock) can destroy the coherence of local time on each slave host. The time elapsed between two events happening on the same host, when enclosing one or several adjustments, cannot be anymore expressed exactly in the host time. The time synchronization process introduces uncertainties and thus an error coming from the surrounding network, mainly because it does not keep track of all the adjustments and of when they were done. In some cases, the order of happening of the events can even appear to be reversed. Additionally, the action of really modifying a clock introduces by itself nondeterministic errors (mainly, the action of adjusting the clock value takes some time we cannot determine, cf. [3]).

The proposed technique uses a continuous function to calculate a master's time and the error made on it, and does not modify local clocks of slaves: events happening on them are recorded with the time read from the original unmodified local clock. For that purpose, it introduces mathematical topology objects, based on the common linear model of computer clocks, to build structures that characterize an observed clock. If local clock modification is required by the application (for example: time-of-day distribution), the result of the conversion function can still be applied to the local clock to adjust it.

These structures allow filtering of observation data with a criterion based on usefulness in improving their global precision. Then, only useless information will be discarded. Since these structures are evolving, a useful information can become useless at a later time. They represent the best information we can extract from the accumulation of observation data and of their associated uncertainty intervals, in relation to the model. This is by nature more precise than statistical information, assumptions, or constraints.

C. *Justification for a Hierarchical Organization*

The topological algorithms proposed in this paper can be used in both groups of time synchronization solutions. However, a hierarchical organization is chosen here because:

1) this kind of solution does not put a constraint on the underlying network (like fully meshed), is simpler to implement, and engenders intrinsically less synchronization messages overhead;
2) it is more appropriate for developing a history about master/slave relationship because of the oriented nature of the relationship;

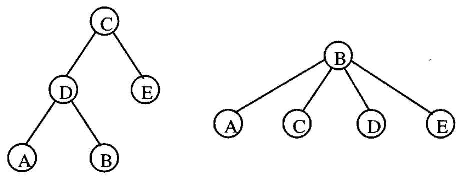

Fig. 1. Examples of hierarchical organization.

3) it is easy to find a common reference time that will be used for time calculations, namely the root of the hierarchical tree;
4) robustness problems can be overcome through diverse techniques, like
    a) in order to pass through *network failures* or *host crashes*, declare any or a number of hosts as references, and slaves who lose their master(s) broadcast then a request for a reference, and choose the one(s) that answers first. A number of policies for reconnecting to an initial master when it is back operational can be developed. This is out of the scope of this paper;
    b) in order to avoid the problem of *false-tickers*, monitor several masters at the same time, and select the best one or combine the best ones. This is also out of the scope here.

Fig. 1 illustrates hosts hierarchically organized in a tree over the network. This organization may be static or evolving over the time. When one wants to measure or order events that have happened on different hosts, the local timestamps of these events are sent upwards over the tree to the nearest common node (the root of the smallest sub-tree containing these hosts). Timestamps and error intervals are converted into master's time along the path toward the common root. Thus, comparisons and delay measurements can be done on this single node in this single time scale with correct and accurate uncertainty intervals.

When one wants to generate synchronized or ordered distributed actions from different hosts, the commands are sent from the root node with the root times of their future occurrence. These times are converted back into the local time along their way to the target nodes, where they will be used to schedule the desired actions. If along the conversion path, the error interval becomes larger than some requested value, the action can then be aborted.

D. Paper Organization

Section II describes the computer clock model that is used for the conversion function. Then in Section III, the time synchronization process is decomposed in two independent levels. Topological structures used for building the conversion function are situated in Level 2, together with features to overcome "accidents." Section IV explains the use by the conversion function of the structures built at Level 2, to produce estimated times and error intervals. Finally, experimental results from a real implementation are presented in Section V.

II. Model of Relation Between Computer Clocks

From a hardware point of view, a computer clock is composed of a counter and a frequency device that controls the increment of the counter. Each tick generated by the frequency device increments the counter by a small amount, which represents the granularity of the clock. For a given clock, the frequency device (generally a quartz) has a frequency that can change, but only slowly with time. It is then realistic to assume it is a constant or a slowly-changing function of time (cf. [3], [6], and [7]).

A conversion law from a clock on host$_a$ to another on host$_b$ only has to know the shift in frequency σ (drift) between these two clocks, and an offset δ at some moment, for example at time 0 on host$_a$. This can be expressed as

$$\boxed{T_b(t) = F_{ab}(T_a(t)) = (1+\sigma).T_a(t) + \delta} \qquad (1)$$

where t is the universal absolute time, and $T_a(t)$ is the time shown by local clock of host$_a$ at that time. The proposed algorithms in this paper do not require σ and δ to be fixed, they only assume that they are slowly changing functions of time.

As said before, host$_a$ cannot fully reconstruct the clock of host$_b$; there are uncertainties. In this context, the following is defined:

"Synchronized" means that for any slave a with master b, there is a conversion function $F_{ab}(\,)$ such that it transforms time of host$_a$ in time of host$_b$ and that the calculated error interval provided with the result is guaranteed and is within a requested maximum interval $[-\eta, \eta]$.

Formally: if $T_a(t)$ and $T_b(t)$ represent the time of host$_a$ and host$_b$, respectively, at universal absolute time t, host$_a$ and host$_b$ are synchronized during time interval $[t_1, t_2]$ if, and only if, there exists $F_{ab}(\,)$ as per (1) such that

$$|F_{ab}(T_a(t)) - T_b(t)| \leq \eta \qquad \forall t/t \in [t_1, t_2].$$

III. Time Synchronization Process

The proposed time synchronization process is split into two levels that are run in each slave host. A master has to know about its slaves to pass them commands or to collect information, and only has to serve the poll requests from slaves by returning time information in its answers. Time synchronization is completely handled by slaves, and these are responsible for guaranteeing the requested precision η. This kind of structure is retrieved in NTP [7], for example.

Level 1 is dedicated to get over the network observation data at one instant from the master clock, and it is responsible for formatting it and estimating accurately the uncertainty interval that is associated with it. It uses some information provided by Layer 2, such as $\sigma_{\min}$ and $\sigma_{\max}$ surrounding the actual value of the drift between local and master clocks. It then gives the whole to Level 2 as a triplet $(t_a, t_b, \varepsilon_b)$, which can be interpreted as: "when the clock at host$_a$ indicates t_a, the clock in host$_b$ indicates a value somewhere in $[t_b - \varepsilon_b, t_b + \varepsilon_b]$." This paper uses a common algorithm for getting observation data, but refines the assessment of the error interval to a further point. Any other algorithm able to provide such a triplet of information can be used instead.

Level 2 is in charge of calling Level 1 to get triplets, of maintaining and filtering a history of the triplets, of evaluating the conversion function parameters, and of determining when it should get observation data through Level 1 to guarantee the requested precision η.

The requested precision η has a lower limit that depends on the granularity of host clocks (that is, the time increment at each clock tick), and on the underlying communication protocol and network used [1]. In this paper, the latter part can be reduced to the minimum transmission round-trip delay experienced for the synchronization messages exchanged between hosts.

One advantage of using a conversion function is that it can estimate times and calculate associated error intervals either in the past before the time synchronization process was started, or in the future. Thus, time synchronization is not limited to the observed period, although best results are of course achieved in this interval. Another advantage is that, since the conversion function is evolving and its precision is monotonously increasing, a conversion that does not provide sufficient precision can be redone at a later time with a better precision.

A. Level 1 of the Time Synchronization Process

The Level 1 mechanism attempts to determine the current offset between the local clock and the master clock, and the uncertainty on this value. It is based on the exchange of two messages as per Fig. 2, conveying time information between the two hosts. The principle is the same as for NTP [7], which is by itself a variant of the returnable-time system used in some digital telephone networks [1]. It is designed to deduce bounds on the offset to apply to the time information from another host it conveys, keeping in mind that it is impossible to exactly determine the transmission delay of a message.

The poll mechanism is an exchange of two messages conveying time information $T_i(t_j)$ defined as per Fig. 2. With the data $T_a(t_1)$, $T_b(t_2)$, $T_b(t_3)$, and $T_a(t_4)$ collected through this mechanism by host$_a$, it is possible to determine the current offset with an uncertainty which is function of the round-trip delay $\mathrm{RT}_a = \varepsilon_a^{1\to2} + \varepsilon_a^{3\to4}$ expressed in the time of host$_a$. $\varepsilon_a^{1\to2}$ and $\varepsilon_a^{3\to4}$ cannot physically be determined, but as shown in Appendix I-A, if σ was exactly known, we could express RT_a as

$$\boxed{\mathrm{RT}_a = (T_a(t_4) - T_a(t_1)) - \frac{T_b(t_3) - T_b(t_2)}{1+\sigma}}. \qquad (2)$$

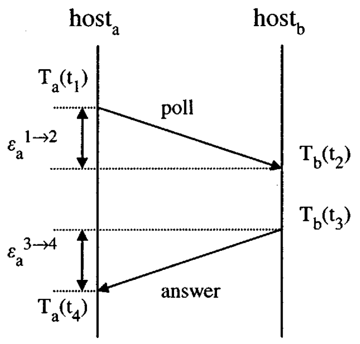

Fig. 2. Poll mechanism.

Then, as per Appendix I-B, the triplet $(t_a, t_b, \varepsilon_b)$ to remember in the history would be

$$\boxed{\left(\frac{T_a(t_1)+T_a(t_4)}{2}, \frac{T_b(t_2)+T_b(t_3)}{2}, \frac{\mathrm{RT}_a}{2}\cdot(1+\sigma)\right)}. \qquad (3)$$

Due to uncertainties, an exact value for σ is not available and a $[\sigma_{\min}, \sigma_{\max}]$ interval must be used instead. As a consequence, the triplet to enter in the history at the moment where the Level 1 is run should be as shown in (4), shown at the bottom of the page; see Appendix I-C for details.

When Level 1 is run for the first time, there is no value yet calculated by Level 2 for $\sigma_{\min}$ and $\sigma_{\max}$. Some initial values have then to be supplied.

It can be seen from calculations in Appendix I-B that exact offset determination is achieved when $\varepsilon_a^{1\to2} = \varepsilon_a^{3\to4}$. This is when both paths of the poll mechanism are symmetric. There is no way however to determine when this happens. Inversely, maximum error is introduced when the paths are totally asymmetric, that is when one of the travel times tends toward zero.

As a result, there is an interest in using a protocol as symmetric as possible and in having the same message length on both directions for the poll mechanism, so that the actual point is best centered in the uncertainty interval $[t_b - \Delta_b, t_b + \Delta_b]$. Moreover, in order to limit the width of the uncertainty interval, the interest is also to minimize $T_b(t_3) - T_b(t_2)$. This means replying as quickly as possible in host$_b$ to the incoming messages of the poll mechanism.

1) Impact of Clock Granularity: The primary effect of clock granularity when reading the value t from a clock is that the

$$\boxed{\begin{array}{l} \left(\dfrac{T_a(t_1)+T_a(t_4)}{2}, \dfrac{T_b(t_2)+T_b(t_3)}{2}, \Delta_b\right) \\ \text{with} \\ \Delta_b = \max\left(\begin{array}{l} \left|(1+\sigma_{\min})\cdot\dfrac{T_a(t_4)-T_a(t_1)}{2} - \dfrac{T_b(t_3)-T_b(t_2)}{2}\right| \\ \left|(1+\sigma_{\max})\cdot\dfrac{T_a(t_4)-T_a(t_1)}{2} - \dfrac{T_b(t_3)-T_b(t_2)}{2}\right| \end{array}\right) \end{array}} \qquad (4)$$

real-valued time is in fact somewhere in $[t,\, t+\tau[$, where τ is the value of the clock increment (the notation $[x,\, y[$ means: any real value between x, included, and y, not included). This has a nonnegligible impact on the uncertainty associated with a measurement by the polling mechanism. Indeed, it is frequent to have increments of 10 or 20 ms for a clock, while the round-trip delay RT_a can be for example of the order of 10 ms on LAN's.

Calculations with clock granularity are detailed in Appendix I-D. The triplet to return to Level 2 accounting for slave and master clocks granularities is then

$$\boxed{\begin{gathered}\left(\frac{T_a(t_1)+T_a(t_4)}{2}+\frac{\tau_a}{2},\ \frac{T_b(t_2)+T_b(t_3)}{2}+\frac{\tau_b}{2},\ \Delta_b\right) \text{ with} \\ \Delta_b=\Delta_{b,\,\mathrm{gran}}+\max(|1+\sigma_{\min}|,\,|1+\sigma_{\max}|)\cdot\frac{\tau_a}{2}+\frac{\tau_b}{2} \\ \text{and } \Delta_{b,\,\mathrm{gran}} \text{ defined as in Appendix I-D.}\end{gathered}} \tag{5}$$

B. Level 2 of the Time Synchronization Process

This level is responsible for calculating the conversion function and for guaranteeing the required precision level. It should get more data through Level 1 when needed, while limiting the number of exchanged synchronization messages to the minimum. To do so, it calculates the interval in the time of host$_a$ where this required precision is respected, at each synchronization sequence. Before the end of that interval, a new synchronization sequence should be executed.

The following list of actions describes the synchronization sequence and the Level 2 process.

1) Get one observation data by calling Level 1. If, current master clock cannot be reached, find a new one and restart. Else, add new data to history.
2) Run the synchronization algorithm on the history to calculate the conversion function and the bounding structures.
3) Calculate the next time a new observation data should be got from master clock to guarantee the requested precision. Rerun the sequence at that time.

The general problem faced here is, how to reconstruct a continuous function giving as result a point and an uncertainty interval, from a linear model (1), and from a set of discrete sampling points with their associated uncertainty intervals (the history of triplets).

Classical approaches use statistics, usually with linear regression or midpoint functions. The algorithm described here builds mathematical structures on the elements of the history. These are used to extract parameters for the conversion function and to calculate the uncertainty interval when a conversion is done.

1) Outline of the Algorithm Used for Extracting Bounding Structures and Conversion Function Parameters: This algorithm is applied to the history H of triplets received from Level 1. H is a set of triplets of the form $H=\{(t_{a,1},\, t_{b,1},\, \Delta_{b,1}),\, (t_{a,2},\, t_{b,2},\, \Delta_{b,2}),\, \cdots,\, (t_{a,N},\, t_{b,N},\, \Delta_{b,N})\}$. It represents the current set of observation data with their uncertainty intervals.

The primary goal of the algorithm is to build bounding structures that will be used for accurate error assessment when a conversion is done, and to place the conversion function between them. The secondary roles are to filter out the contents of H so that it does not become too big, and to detect "accidents" and recover from them.

This algorithm is composed of the following steps.

1) Calculate two sets of points, one made of maximum positions of observation data in their error interval, and one of minimum positions.
2) Build the convex closures of each one of these sets.
3) Detect abnormal situations (crossing closures), and take corrective action by discarding points from history.
4) Remove useless data from history (first step), not used by at least one of the convex closures.
5) Calculate infinite lines with the maximum and minimum gradient for the observed clock, using convex closures.
6) Remove useless data from history (second step), that are made obsolete by the preceding step.
7) Choose conversion function, with regards to convex closures positions, and infinite lines of step 5.

2) Introducing Convex Closures: A definition of the convex closure of a set of points in a plane is given in Appendix II. It is highly recommended to look at that short section before continuing if the notion is not familiar to the reader.

The interesting property of convex closures is that they mathematically define the intuitive notions of "boundary" and "exterior" of a set of points. Any infinite line of the plane whose intersection with the convex closure C ($=$ boundary) of a set of points S is

1) empty ($\emptyset$),
2) reduced to one point,
3) or a segment of C (i.e., the line is tangent to a segment of C)

is said to be at the "exterior" of S (it does not pass through S, or there are not points of S on both of its sides). This notion will be used to define areas where the infinite line representing the conversion function as per (1) cannot be.

In the following, the notions of superior and inferior parts of a convex closure C of S are also used. In a reference plane where points are defined by coordinates (x, y), these will, respectively, designate the parts of C that are "above" and "below" S, with regards to the Y axis. A more formal and mathematical definition is given in Appendix II. To have a quick and intuitive understanding, see also in Appendix II an example with a diagram.

3) Detailed Description of the Algorithm: Fig. 3 depicts a representation in the plane of the observation data in history H, of the bounding structures extracted from them, and of the chosen conversion function. The algorithm is composed of the following steps.

a) If there are$N \geq 2$ points in the history H:

i) *Sets of maximum and minimum positions*: Build two sets of points: $S_U = \{(t_{a,k},\, t_{b,k}+\Delta_{b,k})/k\in\{1,\cdots,N\}\}$, and $S_L=\{(t_{a,k},\, t_{b,k}-\Delta_{b,k})/k\in\{1,\cdots,N\}\}$. Since the uncertainty intervals $[t_{b,k}-\Delta_{b,k},\, t_{b,k}+\Delta_{b,k}]$ are calculated by (5) of Level 1 so that the actual point cannot be outside, S_U is the set of upper possible positions for each observation data in H, and S_L is the set of lower possible positions. This inclusion condition is very

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase® Smarter legal research.