Concurrent Event Detection for Asynchronous Consistency Checking of Pervasive Context

Yu Huang[1,2], Xiaoxing Ma[1,2], Jiannong Cao[3], Xianping Tao[1,2], Jian Lu[1,2]
[1]State Key Laboratory for Novel Software Technology
Nanjing University, Nanjing 210093, China
[2]Department of Computer Science and Technology
Nanjing University, Nanjing 210093, China
{yuhuang, xxm, txp, lj}@nju.edu.cn
[3]Internet and Mobile Computing Lab, Department of Computing
Hong Kong Polytechnic University, Hong Kong, China
csjcao@comp.polyu.edu.hk

***Abstract*—Contexts, the pieces of information that capture the characteristics of computing environments, are often inconsistent in the dynamic and uncertain pervasive computing environments. Various schemes have been proposed to check context consistency for pervasive applications. However, existing schemes implicitly assume that the contexts being checked belong to the same snapshot of time. This limitation makes existing schemes do not work in pervasive computing environments, which are characterized by the asynchronous coordination among computing devices. The main challenge imposed on context consistency checking by asynchronous environments is how to interpret and detect concurrent events. To this end, we propose in this paper the Concurrent Events Detection for Asynchronous consistency checking (CEDA) algorithm. An analytical model, together with corresponding numerical results, is derived to study the performance of CEDA. We also conduct extensive experimental evaluation to investigate whether CEDA is desirable for context-aware applications. Both theoretical analysis and experimental evaluation show that CEDA accurately detects concurrent events in time in asynchronous pervasive computing environments, even with dynamic changes in message delay, duration of events and error rate of context collection.**

I. Introduction

Pervasive computing creates environments that embed computation and communication in a way that organically interacts with humans to enhance or ease their daily behavior [1], [2], [3], [4], [5], [6]. Contexts refer to the pieces of information that capture the characteristics of computing environments, and context-awareness allows applications to intelligently adapt to the dynamic and uncertain pervasive computing environments [7], [8], [9], [10], [11].

Context-aware applications need to monitor whether contexts bear specified property, in order to adapt their behavior accordingly. For example, in a smart office application, the user may specify a constraint on property of the contexts C_1*: Location of the user is the lecture room and a presentation is going on in the lecture room.* Thus the application can adaptively turn mobile phone of the user to vibration mode when C_1 is satisfied. Users may also specify constraints on property of contexts to make sure that the application works correctly, satisfying users' requirements. For example, in a smart digital home application[12], the user may specify that C_2*: the occupant cannot have a shower when the gas-oven is on* for safety purposes. He may also specify that C_3*: lights in the bed room should be off when the occupant is in the living room*, to prevent waste of electricity. Moreover, contexts are often error-prone due to noises [4], [13], [14]. Users may specify constraints to ensure the accuracy of contexts. For example, users may specify that C_4*: no one appears in two different places at the same time*, to make sure that the location context is correctly collected.

Due to the discussions above, the checking of specified constraint on property of contexts, or *context consistency checking*, has been recognized as an important issue [13], [14], [15], [16], [17]. Existing consistency checking schemes are mainly centralized, in which contexts are collected to a central checker. Moreover, it is implicitly assumed that the contexts being checked belong to the same snapshot of time [13], [14], [15], [16]. Refer to the examples above. If the checker collects two pieces of location context: *the user is in the living room* and *the user is in the kitchen*. Only with the assumption that these two pieces of context being checked took place at the same time, can the checker detect that constraint C_4 is violated. We can also see that similar assumptions must hold for the constraint C_1, C_2 and C_3. However, such assumptions do not necessarily hold in pervasive computing environments. Computing devices embedded in the environment usually coordinate in a fully distributed manner, due to the lack of central coordination. They do not have a global cock or any shared memory. Moreover, wireless communications suffer from finite but unbounded delay. Thus, we argue that pervasive computing applications should be modeled as asynchronous distributed systems [18], [19], [20], [21], [22].

The primary challenge imposed on context consistency checking by asynchronous environments is how to interpret and detect concurrent events, which is the main topic of this paper. Specifically, the contribution of this paper is three-fold:

- We propose the *Concurrent Event Detection for Asynchronous consistency checking* (CEDA) algorithm, which detects concurrent events based on the *happen-before*

978-1-4244-3304-9/09/$25.00 ©2009 IEEE

relationship [18], [19] in asynchronous environments.
- An analytical model, together with corresponding numerical results, is derived to study whether CEDA correctly detects concurrent events in time.
- We conduct extensive experimental evaluation to further investigate whether CEDA is desirable for context-aware applications.

Both theoretical analysis and experimental evaluation show that CEDA accurately detects concurrent events in time for context consistency checking in asynchronous pervasive computing environments. CEDA achieves correct and timly detection even when faced with dynamic changes in the message delay, duration of events and error rate of context collection.

The rest of this paper is organized as follows. Section II provides an overview of the existing work. Section III presents design of CEDA. An analytical model is proposed in Section IV, and Section V presents the experimental evaluation. Section VI concludes the paper with a summary and the future work.

II. Related Work

Various schemes have been proposed for context consistency checking in the existing literature. In [13], consistency constraints were modeled by tuples, and consistency checking was based on comparison among elements in the tuples. In [14], consistency constraints were expressed in first-order-logic, and an incremental consistency checking algorithm was proposed. In [15], [16], an ontology was proposed to model context, and consistency constraints were expressed by assertions. Existing schemes for context consistency checking are mainly centralized. Temporal relationships among the collected contexts are not sufficiently considered. It is implicitly assumed that the contexts being checked belong to the same snapshot of time. Such limitations make these schemes do not work in asynchronous pervasive computing environments.

In asynchronous distributed systems, the concept of temporal ordering of events must be carefully reexamined [18]. In the seminal work of Lamport, causality was used to define the *happen-before* relationship between events [18]. Based on the happen-before relationship, the logic clock for asynchronous environments could be devised [23], [24]. In [25], Cooper et al. investigated the detection of general constraints (expressed by logic predicates), which brought combinatorial explosion of the state space. Conjunctive predicates play a key role in specifying consistency constraints, and both weak and strong conjunctive predicates were studied in [21], [22]. However, existing schemes only passively detect predicates, while CEDA makes computing devices proactively send messages to each other, to build temporal ordering among events. Moreover, it was not considered whether constraints were detected correctly and timely in the existing work, but these issues are quite important for context-aware applications. We conduct both theoretical analysis and experimental evaluation to address these issues for CEDA.

TABLE I
Notations in design of CEDA

Notation	Explanation
n	number of normal processes
P_i	the i^{th} normal process involved in the concurrent event detection ($1 \leq i \leq n$)
P_{che}	checker process
LE_i	local event on P_i
lo	beginning of an interval
hi	end of an interval
$VC_i[1..n]$	vector clock on P_i
$flag\_msg\_act$	boolean value used to denote whether there have been new message activities
Que_i	message queue for P_i on P_{che}
Num_{null}	number of queues which have $[lo, null]$ as their head elements

III. Concurrent Event Detection for Asynchronous Consistency Checking

Details of the proposed *Concurrent Events Detection for Asynchronous consistency checking* (CEDA) algorithm are presented in this section. We first describe the system model. Then we present design of CEDA, followed by discussions.

A. System Model

We model the pervasive computing application as a loosely coupled message-passing system without any global clock or shared memory. Communications suffer from finite but unbounded message delay. We assume that no messages are lost, altered, or spuriously introduced. We do not assume that the underlying communication channel is fist-in-first-out(FIFO). Justifications for the assumptions are discussed in Section III.D. A pervasive application consists of a collection of processes, among which n *normal processes* (denoted by $P_1, \cdots, P_n$) are involved in the concurrent event detection. Another *checker process* (denoted by P_{che}) is dedicated for checking of concurrent events among the normal processes.

Modeling events by intervals. We model the local event on P_i by boolean predicate LE_i. LE_i is true if the local event is taking place on P_i. Otherwise, it is changed to false. For concurrent event detection, we need to detect whether local events $LE_1, LE_2, \cdots, LE_n$ are concurrently true. We record the intervals in which $LE_i = true$. The false-to-true and the true-to-false transitions (denoted by $\uparrow$ and $\downarrow$ respectively) of LE_i correspond to the beginning and end of the interval (denoted by lo and hi respectively).

Temporal ordering between events. CEDA detects concurrent events based on the *happen-before* relationship as defined in [18], [21], [22], and adopts the vector clock used in [22]. Each normal process P_j keeps VC_j, its own local vector of timestamps. $VC_j[i]$ ($i \neq j$) is ID of the last message from P_i, which has a causal relationship to P_j. $VC_j[j]$ for P_j is the next message id P_j will use. As discussed in [22], [23], [24], the happen-before relationship between events can be simply checked by comparing their vector clock timestamps. When the events $\uparrow$ and $\downarrow$ take place, VC_i is recorded as lo and hi of the interval respectively.

Notations used in design of CEDA are listed in Table I.

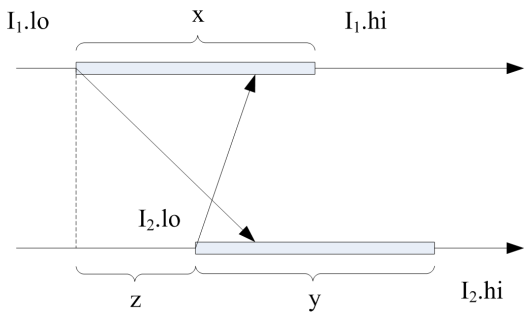

Fig. 1. Overlapping intervals

B. CEDA on the Normal Process Side

The fact that n intervals $I_1, I_2, \cdots, I_n$ overlap is equivalent to:

$$(I_j.lo \rightarrow I_k.hi) \wedge (I_k.lo \rightarrow I_j.hi), \forall 1 \leq j \neq k \leq n \quad (1)$$

The case where two intervals overlap is shown in Fig 1.

On the normal process side, each time $LE_i \uparrow$, VC_i is recorded as a lo, and sent to each P_j $(j \neq i)$. Sending messages among normal processes helps to build temporal ordering among los and his of the intervals, as required by Formula (1). P_i also sends the incomplete interval $[lo, null]$ to P_{che}. Each time $LE_i \downarrow$, VC_i is recorded as a hi, and the incomplete interval $[null, hi]$ is sent to P_{che}.

Note that P_i is not required to send VC_i every time LE_i changes. P_i does not need to send the timestamp if there has been no message activity since the last time the timestamp was sent. This is because the vector clock can change its value only when a message is sent or received, as discussed in Section III.A. A boolean variable $flag\_msg\_act$ is used to record whether there has been any message activity. The initial value of $flag\_msg\_act$ is true. Pseudo codes of the CEDA algorithm on the normal process side are listed in Algorithm 1.

C. CEDA on the Checker Process Side

P_{che} receives $[lo, null]$ and $[null, hi]$ from each P_i. A $[lo, null]$ and the corresponding $[null, hi]$ can be combined to obtain a complete interval $[lo, hi]$. A separate queue of intervals (Que_i) is maintained for each P_i. P_{che} executes Algorithm 2 to compare the received los and his. Head element of any queue, whose hi is not greater than lo of head elements of all other queues are deleted. It is because CEDA will not be able to build the relationship expressed in Formula (1) with such an interval. Thus they should be eliminated for further comparison. For complete intervals (neither lo nor hi is null), CEDA detects whether they overlap by comparing timestamps of their los and his [22]. For incomplete intervals, we use Num_{null} to denote the number of queues which have $[lo, null]$ as their head elements. Observe that CEDA has chances to detect concurrent events when $Num_{null} = 1$. This is called an *early detection*, which is analyzed in detail in Section IV.B.

Algorithm 1 CEDA on P_i

```
1: Upon LE_i ↑
2: send([VC_i]) to each P_j (j ≠ i);
3: if flag_msg_act then
4:    send([lo, null]_i) to P_che;
5:    VC_i[i] + +;
6: else
7:    store [lo, null] in the message queue;
8: end if
9:
10: Upon LE_i ↓
11: if flag_msg_act then
12:    if there is [lo, null] in the message queue then
13:       retrieve the stored timestamp and send([lo, hi]) to
          P_che;
14:    else
15:       send([null, hi]) to P_che;
16:    end if
17:    VC_i[i] + +;
18:    flag_msg_act = false;
19: else
20:    clear the message queue;
21: end if
22:
23: Upon receive msg(VC_j)
24: for k = 1 to n do
25:    VC_i[k] = max{VC_i[k], VC_j[k]};
26: end for
27: flag_msg_act = true;
```

We assume that P_{che} receives messages from normal processes in FIFO order as in [21], [22]. Note that this is not a restrictive assumption. We do not require FIFO for the underlying communication. P_{che} needs to implement the FIFO property for efficiency purposes. If the underlying communication is not FIFO, P_{che} ensures this property by using sequence numbers in messages.

D. Discussion

Complexity. On the checker process side, the number of comparisons is $O(n^2p)$, where n is the number of queues each of length at most p. On the normal process side, the number of message activities is $O(p)$. The space complexity is $O(n)$, for recording the vector clock. Note that the message complexity of CEDA is mainly due to building the happen-before relationship between los and his when there is not any global clock. Existing work may impose less message complexity. However, they cannot work in asynchronous pervasive computing environments.

Justifications for the assumptions. We assumed the reliability of message passing in Section III.A. Note that even with these assumptions, we cannot always guarantee correct detection of concurrent events (as discussed in Section IV). Without message reliability, we only need to revise our algorithm to discard the incomplete information on P_{che} due to the message

Algorithm 2 CEDA on P_{che}

Upon receive($[lo, null]_k$)
enqueue $[lo, null]_k$ in Que_k;
if $[lo, null]_k \neq head(Que_k)$ **then**
  return;
end if
for $i = 1$ to $n, i \neq k$ **do**
  while $\neg(head(Que_k).lo \rightarrow head(Que_i).hi)$ **do**
    delete_head(Que_i);
  end while
end for
if $Num_{null} = 1$ **then**
  if $\forall i \neq k : head(Que_i).lo \rightarrow head(Que_k).lo \wedge head(Que_k).lo \rightarrow head(Que_i).hi$ **then**
    return early detection;
  end if
end if

Upon receive $[null, hi]_k$ or $[lo, hi]_k$
combine $[null, hi]_k$ with $[lo, null]_k$, obtaining $[lo, hi]_k$;
if $[lo, hi]_k \neq head(Que_k)$ **then**
  return;
end if
$changed := \{k\}$;
while $changed \neq \emptyset$ **do**
  $newchanged := \emptyset$;
  for $i \in changed, 1 \leq j \leq n$ **do**
    if $\neg(head(Que_j).lo \rightarrow head(Que_i).hi)$ **then**
      $newchanged := newchanged \cup \{i\}$;
    end if
    if $\neg(head(Que_i).lo \rightarrow head(Que_j).hi)$ **then**
      $newchanged := newchanged \cup \{j\}$;
    end if
  end for
  $changed := newchanged$;
  for each i in $changed$ **do**
    delete_head(Que_i);
  end for
end while
if $Num_{null} = 1$ and $head(Que_l) = [lo, null]$ **then**
  if $\forall i \neq l : head(Que_i).lo \rightarrow [lo, null]_l.lo \wedge [lo, null].lo \rightarrow head(Que_i).hi$ **then**
    return early detection;
  end if
end if
if $Num_{null} = 0 \wedge \forall i : Que_i \neq \emptyset$ **then**
  return concurrent events detected;
end if

TABLE II
NOTATIONS IN ANALYSIS OF CEDA

Notation	Explanation
P_{corr}	probability of correct detection
P_{early}	probability of early detection
X, Y, Z	random variables depicting the temporal attributes of intervals
$F_1(X)$	probability functions of X
$F_2(Y)$	probability functions of Y
$F_3(Z)$	probability functions of Z
λ	message delay rate
λ_i	Poisson arrival rate of intervals
μ_i	rate of interval duration

loss. The rationale for concurrent event detection still remains the same. It is obvious that the probability of correct detection will further decrease when faced with message loss.

IV. PERFORMANCE ANALYSIS

In the previous Section III, we presented the design of CEDA. However, does CEDA work in pervasive computing environments? Specifically, can CEDA achieve accurate detection of concurrent events? Can CEDA detect concurrent events in time? We investigate these issues by theoretical analysis in this section. In the following Section V, we further evaluate CEDA by experiments. In our analysis, we first focus on the case where the number of normal processes is 2. Then we discuss the case with n (≥ 3) normal processes. Notations used in the analysis are listed in Table I and Table II.

A. Probability of Correct Detection

It is not possible to guarantee accurate detection of concurrent events in asynchronous systems [26], [27]. Formally, there does not exist an algorithm $\mathcal{A}$, which always achieves correct detection of concurrent events. Thus we study the probability that CEDA achieves correct detection. Toward this objective, we first investigate when and how CEDA falsely detects concurrent events.

Observe that CEDA is never false positive, i.e., when CEDA detects overlapping intervals (of concurrent events), the intervals definitely overlap. This is because CEDA detects overlapping intervals only when it guarantees that Formula (1) is satisfied. However, CEDA could be false negative, i.e., when CEDA does not detect overlapping intervals, it is possible that the intervals do overlap actually. For example, two intervals do overlap as shown in Fig. 2, but if I_1 receives the message from I_2 (denoted by $msg(I_2)$) later than $I_1.hi$, CEDA will not detect that I_1 and I_2 overlap.

CEDA correctly detects overlapping intervals if and only if $msg(I_1)$ and $msg(I_2)$ are received in time. Formally, the probability of correct detection is:

$$P_{corr} = P(x, y, z) = Pr\{msg(I_1).delay \leq y + z\} \\ Pr\{msg(I_2).delay \leq x - z\} \quad (2)$$

We define random variables X, Y and Z, whose values are x, y and z respectively, as shown in Fig. 2. The expected

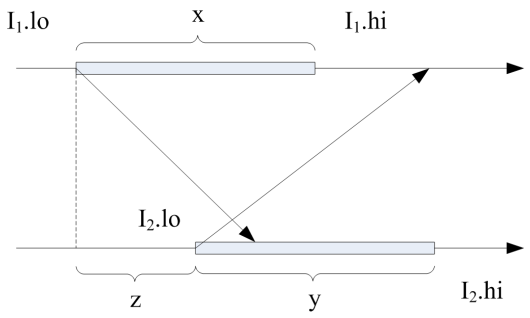

Fig. 2. CEDA being false negative

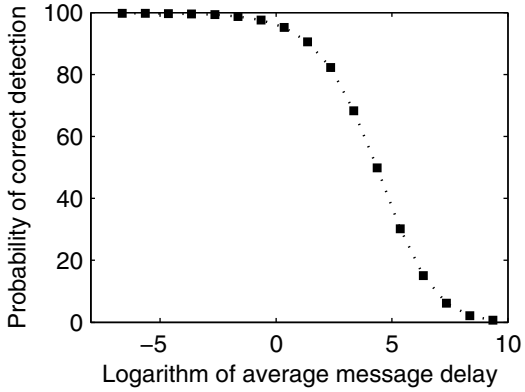

Fig. 3. Probability of correct detection

probability of correct detection is:

$$E[P_{corr}] = E[P(x,y,z)] = \int_0^\infty \{\int_0^\infty [\int_0^x P(x,y,z) \, dF_3(z)] dF_2(y)\} dF_1(x) \quad (3)$$

Here, the probability functions $F_1(x) = Pr\{X \leq x\}$, $F_2(y) = \{Y \leq y\}$, $F_3(z) = Pr\{Z \leq z\}$.

To further investigate $E[P_{corr}]$, we model intervals on P_i based on the queuing theory [28], [29]. Specifically, a queue of intervals with Poisson arrival rate λ_i is adopted. The duration of intervals follows the exponential distribution of rate $\mu_i(\frac{\lambda_i}{\mu_i} \leq 1)$. We also need to model the message delay, which greatly affects $E[P_{corr}]$. The distribution of message delay is affected by implementation of the underlying layers of network (e.g., the MAC or routing layer), and greatly varies in different scenarios. Although it is doubted whether there exists a universal model of message delay, the exponential distribution is widely used and also evaluated by both simulations and experiments [30], [31]. In our analysis, we adopt the exponential distribution to model message delay. Note that our analysis is also applicable when message delay follows other types of distributions. With the models we adopt, the probability of correct detection is:

$$P_{corr} = P(x,y,z) = (1 - e^{-\lambda(y+z)})(1 - e^{-\lambda(x-z)}) \quad (4)$$

According to [29], the probability function of the interval length is:

$$\begin{aligned} F_1(x) &= Pr\{X \leq x\} \\ &= \int_0^x \sum_{k=1}^{\infty} \frac{\mu_1(\lambda_1\mu_1 x^2)^{k-1}}{k!(k-1)!} e^{-(\lambda_1+\mu_1)x} dx \end{aligned}$$

$$\begin{aligned} F_2(y) &= Pr\{Y \leq y\} \\ &= \int_0^y \sum_{k=1}^{\infty} \frac{\mu_2(\lambda_2\mu_2 y^2)^{k-1}}{k!(k-1)!} e^{-(\lambda_2+\mu_2)y} dy \end{aligned}$$

Since the arrival of I_2 follows the Poisson process, we have that, for $z > 0$:

$$F_3(z) = Pr\{Z \leq z\} = 1 - e^{-\lambda_2 z}$$

To illustrate performance of CEDA in terms of the probability of correct detection ($E[P_{corr}]$), we change the average message delay and obtain the numerical results, as shown in Fig. 3. We choose to vary the average message delay because it has the most significant impact on $E[P_{corr}]$. According to the numerical results, we find that $E[P_{corr}]$ remains near 100% when the message delay is not quite long. As the average message delay significantly increases, $E[P_{corr}]$ decreases quite slowly (note that the x-axis is the logarithm of message delay). This ensures that CEDA achieves satisfying probability of correct detection in various pervasive computing environments with different message delay.

B. Probability of Early Detection

In context-aware applications, it is quite desirable, if not a must, to detect concurrent events as early as possible. However, this issue was not sufficiently addressed in the existing work, e.g., in debugging of distributed programs[25], [21], [22]. In existing schemes, P_{che} only receives timestamps of a complete interval (with both lo and hi). Thus P_{che} detects concurrent events no earlier than the latest hi among all the intervals [22]. CEDA detects concurrent events in a finer granularity. The timestamp of lo or hi is sent right after the $\uparrow$ or $\downarrow$ transition of LE_i. Thus, CEDA has chances to detect concurrent events earlier than existing schemes.

We first discuss when and how CEDA achieves early detection. As shown in Fig. 4, if concurrent events are checked only when both lo and hi of the intervals are collected, we detect concurrent events after $I_2.hi$. However, we can still detect that I_1 and I_2 overlap without the timestamp of $I_2.hi$. Observe that if we guarantee:

$$(I_1.lo \rightarrow I_2.lo) \wedge (I_2.lo \rightarrow I_1.hi) \wedge y > x - z \quad (5)$$

we guarantee that Formula (1) holds, thus guaranteeing I_1 and I_2 overlap. In this case, CEDA has chances to detect concurrent events after $I_1.hi$ is collected, not having to wait for $I_2.hi$.

Based on the analysis above, the probability of early detec-

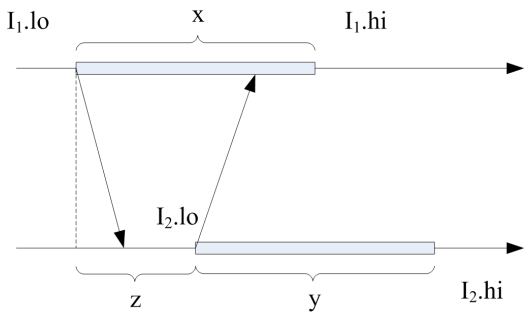

Fig. 4. CEDA achieving early detection

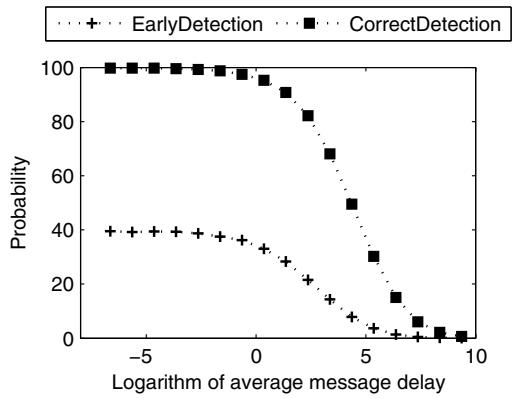

Fig. 5. Probabilities of early detection and correct detection

tion is:

$$\begin{aligned} & P_{early}(x,y,z) \\ &= Pr\{msg(I_1).delay < z\} \\ &\quad Pr\{msg(I_2).delay < x-z\}Pr\{y > x-z\} \\ &= (1-e^{-\lambda z})(1-e^{-\lambda(x-z)})(1-F_2(x-z)) \end{aligned} \quad (6)$$

The expected probability of correct detection is:

$$E[P_{early}(x,y,z)] = \int_0^\infty \{\int_0^\infty [\int_0^x P_{early}(x,y,z) \, dF_3(z)]dF_2(y)\}dF_1(x) \quad (7)$$

We also illustrate performance of CEDA in terms of the probability of early detection ($E[P_{early}]$) by numerical results. From Fig. 5, we find that P_{early} is around half of P_{corr} when the message delay is not quite long and both probabilities remain stable. Both probabilities decrease in a similar way as the average message delay significantly increases, i.e., $E[P_{early}]$ can also tolerate significant increase in the average message delay. Thus CEDA achieves satisfying performance in terms of both probabilities in environments with different message delay.

C. Discussion

Our analytical model can be applied in cases with any number of intervals ($n \geq 2$). In the analysis above, we mainly focus on the case where $n = 2$. For n (≥ 3) intervals, we can obtain $E[P_{corr}]$ in a similar way as in Section IV.A by calculating the probability that any two overlapping intervals are correctly detected. Meanwhile, CEDA may achieve early detection without receiving the latest hi from $Interval_k$. In this case, early detection is achieved when: (i) Formula (5) holds between $Interval_k$ and every $Interval_i (i \neq k)$; (ii) all $Interval_i (i \neq k)$ overlap. Detailed derivation of P_{early} in this case is omitted in this paper.

TABLE III
EXPERIMENT CONFIGURATIONS

Parameter	Value
Number of offices	4
Number of users	30
Lifetime of application	8 h
Duration of stay in office	60 to 960 s
Message delay	0.01 to 655.36 s
RFID error reate	10%, 20%, ⋯, 50%

V. EXPERIMENTAL EVALUATION

In this section, we conduct simulation experiments to further evaluate whether CEDA is desirable for context-aware applications in asynchronous pervasive computing environments. We first describe the experiment setup, and then analyze the evaluation results.

A. Experiment Setup

In the experimental evaluation, we simulate a smart office scenario, in which users move around in different offices. The duration of users' stay in an office follows the exponential distribution. User's location is the context we are interested in, since location contexts have been recognized as the most important type of context and are widely used and studied in applications and existing literature[32], [33]. Users carry RFID tags and user's location context is collected by RFID readers in the offices. To ensure that the location context is correctly collected, users specify the consistency constraint C_4*: no one appears in two different places at the same time*, as discussed in Section I. Location contexts are produced with controlled error rates of 10%, 20%, 30%, 40% and 50%. These rates are based on real-life observations about RFID error rates [34], [35], [36]. We model the message delay by the exponential distribution.

In the evaluation, we study the probabilities of correct detection and early detection. The average message delay, the average duration of a user's stay in the office and error rate of context collection are varied. We choose to vary message delay and duration of a user's stay in the office since they are closely related to the probabilities we are investigating, according to our analytical results in Equation (3) and (7). We vary the error rate of context collection since it is the error in location context, which motivates context consistency checking. Detailed experiment configurations are listed in Table III.

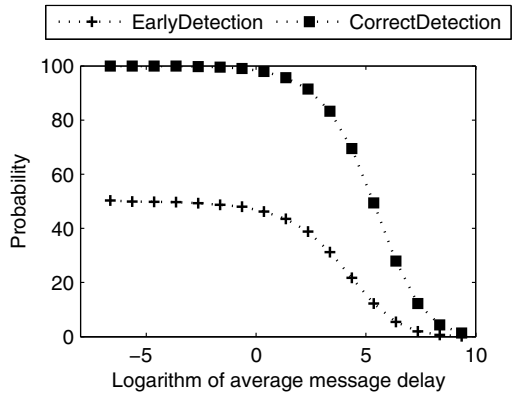

Fig. 6. Effects of tuning the average message delay

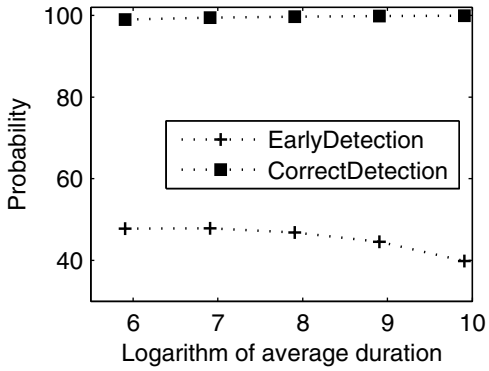

Fig. 7. Effects of tuning the duration of stay in an office

B. Effects of Tuning the Message Delay

In the first experiment, we study how changes in the average message delay affect the probabilities of correct detection ($E[P_{corr}]$) and early detection ($E[P_{early}]$). As shown in Fig. 6, $E[P_{corr}]$ and $E[P_{early}]$ remain stable when the message delay is not quite long. $E[P_{corr}]$ is high (near 100%) and $E[P_{early}]$ is around half of $E[P_{corr}]$. As the average message delay significantly increases, $E[P_{corr}]$ and $E[P_{early}]$ decrease slowly (note that the x-axis is the logarithm of message delay). This ensures that CEDA obtains desirable $E[P_{corr}]$ and $E[P_{early}]$ when employed by context-aware applications in scenarios with different message delay.

Note that the experimental evaluation results in this experiment are in accordance with the numerical results in Fig. 5, except that $E[P_{early}]$ in this experiment is slightly higher. The increase in $E[P_{early}]$ is mainly because we generate the trace of a user in different offices using a simpler model in this experiment. The user just stays in one office for a duration of exponential distribution and then moves to another random office.

C. Effects of Tuning the Duration of Stay in an Office

In the second experiment, we study effects of tuning the average duration of the user's stay in one office. We find that $E[P_{corr}]$ slightly increases, while $E[P_{early}]$ slightly decreases when the duration increases, as shown in Fig. 7. We can explain changes in the probabilities by referring to Equations (4) and (6). As shown in Equation (4), when the duration of the user staying in the office increases, values of both x and y increase, resulting in the increase in P_{corr} and $E[P_{corr}]$. The decrease in P_{early} mainly results from the increase in $F_2(x-z)$, as shown in Equation (6). Since $F_2(x-z)$ is a probability function and is thus monotonically non-decreasing, increase in x results in the increase in $F_2(x-z)$.

Overall, the impact of average duration on $E[P_{corr}]$ and $E[P_{early}]$ is not significant, which makes CEDA widely applicable to detect concurrent events of different durations.

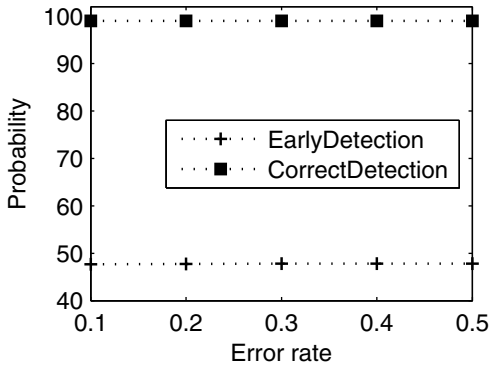

Fig. 8. Effects of tuning the error rate of context collection

D. Effects of Tuning the Error Rate of Context Collection

In the third experiment, we study the effects of tuning the error rate of context collection. Based on Fig. 8, we find that changes in the error rate have little impact on both $E[P_{corr}]$ and $E[P_{early}]$, which can be explained by the expressions of $E[P_{corr}]$ and $E[P_{early}]$ in Equations (3) and (7) respectively.

Note that the increase in erroneous contexts may increase the number of messages transmitted and also increase the load on the checker process. Detailed evaluation such impacts on $E[P_{corr}]$ and $E[P_{early}]$ are out of the scope of this paper.

E. Lessons Learned

Based on the results of experimental evaluation, we show that CEDA is desirable for context-aware applications in asynchronous pervasive computing environments. In particular,

- CEDA can tolerate fairly long message delay and still achieve satisfying probabilities of correct detection and early detection;
- CEDA is not significantly affected by changes in the duration of events being detected;
- CEDA can also tolerates changes in the error rate of context collection, while achieving stable performance in terms of probabilities of correct detection and early detection.

VI. Conclusion and Future Work

In this paper, we study concurrent event detection for consistency checking of pervasive context in asynchronous environments. Toward this objective, our contributions can be described as follows: (1) we propose the CEDA algorithm; (2) an analytical model is derived to investigate performance of CEDA, and we further illustrate our analysis by numerical results; (3) we conduct extensive experimental evaluation to further evaluate CEDA in a variety of pervasive computing environments.

Currently the CEDA algorithm still suffers from several limitations. In our future work, we need to investigate how CEDA can deal with dynamic changes in the number of processes involved in the detection of concurrent events. We also need to study how to further reduce the message complexity and improve bounds of the probabilities of correct detection and early detection. Moreover, we should evaluate CEDA in different scenarios with different types of contexts and consistency constraints.

Acknowledgment

The authors would like to thank Dr. Chang Xu and Chun Cao for their constructive comments and suggestions.

This work is supported by National Grand Fundamental Research 973 Program of China(No.2009CB320702), National Natural Science Foundation of China(No.60736015), National High-Tech Research and Development 863 Program of China(No.2007AA01Z178) and "Climbing" Program of Jiangsu Province, China(No.BK2008017).

References

[1] M. Weiser, "The computer for the 21st century," *Scientific American*, vol. 265, no. 3, pp. 66–75, September 1991.

[2] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste, "Project aura: Toward distraction-free pervasive computing," *IEEE Pervasive Computing*, vol. 1, no. 2, pp. 22–31, 2002.

[3] M. Romn, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt, "A middleware infrastructure for active spaces," *IEEE Pervasive Computing*, vol. 1, no. 4, pp. 74–83, 2002.

[4] M. Garofalakis, K. Brown, and M. Franklin, "Probabilistic data management for pervasive computing: The data furnace project," *IEEE Data Engineering Bulletin*, vol. 29, pp. 57–63, Mar. 2006.

[5] S. Kalasapur, M. Kumar, and B. Shirazi, "Evaluating service oriented architectures (SOA) in pervasive computing," in *Proc. IEEE International Conference on Pervasive Computing and Communications (PERCOM'06)*, Pisa, Italy, Mar. 2006, pp. 276–285.

[6] Y. Huang, J. Cao, Z. Wang, B. Jin, and Y. Feng, "Achieving flexible cache consistency for pervasive internet access," in *Proc. IEEE International Conference on Pervasive Computing and Communications (PERCOM'07)*, New York City, USA, Mar. 2007, pp. 239–250.

[7] A. Dey, G. Abowd, and D. Salber, "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications," *Human-Computer Interaction*, vol. 16, pp. 97–166, Feb. 2001.

[8] K. Henricksen and J. Indulska, "A software engineering framework for context-aware pervasive computing," in *Proc. IEEE International Conference on Pervasive Computing and Communications (PERCOM'04)*, Orlando, Florida, USA, Mar. 2004, pp. 77–86.

[9] T. Gu, H. K. Pung, and J. K. Yao, "Towards a flexible service discovery," *J. Netw. Comput. Appl.*, vol. 28, no. 3, pp. 233–248, 2005.

[10] P. Hu, J. Indulska, and R. Robinson, "An autonomic context management system for pervasive computing," in *Proc. IEEE International Conference on Pervasive Computing and Communications (PERCOM'08)*, Hong Kong, China, Mar. 2008, pp. 213–223.

[11] T. Gu, H. K. Pung, and D. Zhang, "Peer-to-peer context reasoning in pervasive computing environments," in *Proc. IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOMW'08)*, Hong Kong, China, Mar. 2008, pp. 406–411.

[12] R. Harper, *Inside the Smart Home: Ideas, Possibilities and Methods*. London: Springer, 2003.

[13] C. Xu and S. C. Cheung, "Inconsistency detection and resolution for context-aware middleware support," in *Proc. ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'05)*, Lisbon, Portugal, Sep. 2005, pp. 336–345.

[14] C. Xu, S. C. Cheung, and W. K. Chan, "Incremental consistency checking for pervasive context," in *Proc. International Conference on Software Engineering (ICSE'06)*, Shanghai, China, May 2006, pp. 292–301.

[15] Y. Bu, T. Gu, X. Tao, J. Li, S. Chen, and J. Lu, "Managing quality of context in pervasive computing," in *Proc. International Conference on Quality Software (QSIC'06)*, Beijing, China, Oct. 2006, pp. 193–200.

[16] Y. Bu, S. Chen, J. Li, X. Tao, and J. Lu, "Context consistency management using ontology based model," in *Proc. Current Trends in Database Technology (EDBT'06)*, Munich, Germany, Mar. 2006, pp. 741–755.

[17] Y. Huang, X. Ma, X. Tao, J. Cao, and J. Lu, "A probabilistic approach to consistency checking for pervasive context," in *Proc. IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC'08)*, Shanghai, China, Dec. 2008, pp. 387–393.

[18] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[19] R. Schwarz and F. Mattern, "Detecting causal relationships in distributed computations: in search of the holy grail," *Distrib. Comput.*, vol. 7, no. 3, pp. 149–174, 1994.

[20] Özalp Babaoğlu and K. Marzullo, *Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1993.

[21] V. Garg and B. Waldecker, "Detection of weak unstable predicates in distributed programs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, pp. 299–307, Mar. 1994.

[22] V. K. Garg and B. Waldecker, "Detection of strong unstable predicates in distributed programs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, pp. 1323–1333, Dec. 1996.

[23] C. J. Fidge, "Partial orders for parallel debugging," in *Proc. ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging*, Madison, Wisconsin, US, May 1988, pp. 183–194.

[24] F. Mattern, "Virtual time and global states of distributed systems," in *Proc. International Workshop on Parallel and Distributed Algorithms*, Holland, 1989, pp. 215–226.

[25] R. Cooper and K. Marzullo, "Consistent detection of global predicates," in *Proc. ACM/ONR Workshop on Parallel and Distributed Debugging*, New York, NY, USA, 1991, pp. 167–174.

[26] J. Y. Halpern and Y. Moses, "Knowledge and common knowledge in a distributed environment," *J. ACM*, vol. 37, no. 3, pp. 549–587, 1990.

[27] P. Panangaden and K. Taylor, "Concurrent common knowledge: defining agreement for asynchronous systems," *Distrib. Comput.*, vol. 6, no. 2, pp. 73–93, 1992.

[28] B. Bunday, *An Introduction to Queueing Theory*. London: A Hodder Arnold Publication, 1996.

[29] Y. Tang and X. Tang, *Queuing Theory: Fundamentals and Analysis Techniques (in Chinese)*. Beijing, China: Science Press, 2006.

[30] N. Duffield and F. Lo Presti, "Multicast inference of packet delay variance at interior network links," in *Proc. IEEE Conference on Computer Communications (INFOCOM'00)*, Tel Aviv, Israel, Mar 2000, pp. 1351–1360 vol.3.

[31] N. G. Duffield, J. Horowitz, F. L. Presti, and D. F. Towsley, "Network delay tomography from end-to-end unicast measurements," in *Proc. Thyrrhenian International Workshop on Digital Communications (IWDC'01)*, London, UK, 2001, pp. 576–595.

[32] R. Want, A. Hopper, a. Veronica Falc and J. Gibbons, "The active badge location system," *ACM Trans. Inf. Syst.*, vol. 10, no. 1, pp. 91–102, 1992.

[33] L. M. Ni, Y. Liu, Y. C. Lau, and A. P. Patil, "Landmarc: Indoor location sensing using active rfid," in *Proc. IEEE International Conference on Pervasive Computing and Communications (PERCOM'03)*, Dallas, Texas, US, Mar. 2003, pp. 407–415.

[34] C. Xu, S. Cheung, W. Chan, and C. Ye, "Heuristics-based strategies for resolving context inconsistencies in pervasive computing applications,"

in *Proc. International Conference on Distributed Computing Systems (ICDCS'08)*, Beijing, China, Jun. 2008, pp. 713–721.

[35] S. R. Jeffery, M. Garofalakis, and M. J. Franklin, "Adaptive cleaning for rfid data streams," in *Proc. International Conference on Very Large Data Bases (VLDB'06)*, Seoul, Korea, Sep. 2006, pp. 163–174.

[36] J. Rao, S. Doraiswamy, H. Thakkar, and L. S. Colby, "A deferred cleansing method for rfid data analytics," in *Proc. International Conference on Very Large Data Bases (VLDB'06)*, Seoul, Korea, Sep. 2006, pp. 175–186.