Adaptability and Stability in Dynamic Integration of Body Sensor Networks with Clouds

Yi Cheng Ren*, Junichi Suzuki*, Shingo Omura‡ and Ryuichi Hosoya‡
* University of Massachusetts, Boston
Boston, MA 02125-3393, USA
Email: {yiren001,jxs}@cs.umb.edu
‡OGIS International, Inc.
San Mateo, CA 94402, USA
Email: {omura,hosoya}@ogis-international.com

***Abstract*—This paper considers a multi-tier architecture for cloud-integrated body sensor networks (BSNs), called Body-in-the-Cloud (BitC), which is designed for home healthcare with on-body physiological and activity monitoring sensors. This paper formulates an optimization problem to integrate BSNs with a cloud in BitC and approaches the problem with an evolutionary game theoretic algorithm. BitC allows BSNs to adapt their configurations (i.e., sensing intervals) to operational conditions (e.g., data request patterns) with respect to multiple performance objectives such as resource consumption and data yield. BitC theoretically guarantees that each BSN performs an evolutionarily stable configuration strategy, which is an equilibrium solution under given operational conditions. Simulation results verify this theoretical analysis; BSNs seek equilibria to perform adaptive and evolutionarily stable configuration strategies under dynamic changes of operational conditions. BitC outperforms NSGA-III in optimality, stability, convergence speed and execution time.**

***Index Terms*—Body sensor networks, Cloud computing, Multiobjective optimization, Evolutionary algorithms**

I. Introduction

This paper studies an architecture, called Body-in-the-Cloud (BitC), which is designed to integrate body sensor networks (BSNs) with cloud computing platforms for remotely and continuously performing physiological and activity monitoring for homebound patients. A BSN is a wireless network of on/in-body sensors for, for example, heart rate, oxygen saturation and fall detection. BitC virtualizes per-patient BSNs onto clouds by taking advantage of cloud computing features such as scalability in data processing/storage and availability through multi-regional application deployment.

This paper formulates an optimization problem to integrate BSNs with a cloud in BitC by adjusting configuration parameters (e.g., sensing intervals and data transmission intervals) and approaches the problem with BitC's integration optimizer, which exhibits the following properties:

- *Adaptability*: BItC allows BSNs to adapt and optimize configurations according to operational conditions (e.g., data request patterns placed by cloud applications and availability of resources such as bandwidth and memory) with respect to performance objectives such as bandwidth consumption, energy consumption and data yield.
- *Stability*: BitC allows BSNs to seek stable adaptation decisions by minimizing oscillations (or non-deterministic inconsistencies) in decision making. Stability is considered as the reachability to at least one of equilibrium solutions in decision making. A lack of stability results in making inconsistent adaptation decisions in different optimization attempts/trials with the same problem settings.

BitC is designed to attain the adaptability and stability properties with evolutionary computation (EC) and evolutionary game theory (EGT), respectively. BitC leverages EC, particularly an evolutionary multiobjective optimization algorithm (EMOA), because, in general, EMOAs are robust problem-independent search methods that seek optimal solutions (i.e., optimal adaptation decisions) with reasonable computational costs by maintaining a small ratio of search coverage to the entire search space [1]. BitC employs EGT as a means to mathematically formulate adaptive decision making and theoretically guarantee that each decision making process converges to an evolutionarily stable equilibrium where a specific adaptation decision is deterministically made under a particular set of operational conditions [2].

By integrating EC and EGT, BitC provides an EGT-backed EMOA that allows BSNs to (1) seek the solutions to optimally adapt their configurations and (2) operate at equilibria by making evolutionarily stable adaptation decisions. In BitC, each BSN maintains a set (or a population) of configuration strategies (solution candidates), each of which specifies a set of configuration parameters for that BSN. BitC theoretically guarantees that, through a series of evolutionary games between BSN configuration strategies, the population state (i.e., the distribution of strategies) converges to an evolutionarily stable equilibrium regardless of the initial state. (A dominant strategy in the evolutionarily stable population state is called an *evolutionarily stable strategy* (ESS).) Given this theoretical property, BitC allows each BSN to operate at an equilibrium by using an ESS as an adaptive configuration strategy.

This paper describes the design of BitC and evaluates its optimality and stability in making adaptation decisions under dynamic changes of operational conditions. Simulation results demonstrate that BitC allows BSNs to seek equilibria to perform evolutionarily stable configurarion strategies and adapt their configurations to given operational conditions. It outperforms NSGA-III, one of the state-of-the-art EMOAs [3], in optimality, stability, convergence speed and execution time. Under dynamic changes of operational conditions, BitC efficiently reconfigures BSNs by repeating its optimization process based on the history of its prior optimization processes. The notion of dynamic optimization termination allows BitC to gain significant speedup in the execution time of its optimization process.

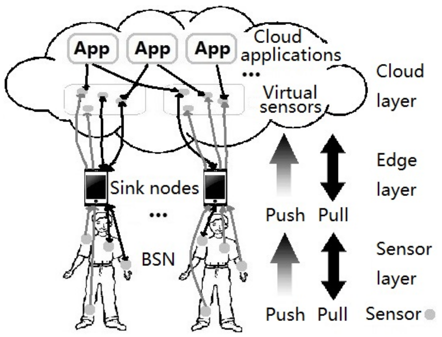


Fig. 1: An Architectural Overview of BitC

II. An Architectural Overview of BitC

BitC consists of the *sensor*, *edge* and *cloud* layers (Fig. 1).

Sensor Layer: operates one or more BSNs on a per-patient basis. Each BSN contains one or more sensor nodes, each of which is equipped with different types of sensors. Sensor nodes are wirelessly connected to a dedicated per-patient device or a patient's computer (e.g., a smartphone or tablet machine) that serves as a *sink* node. This paper assumes the star topology among a sink node and sensor nodes. Each sensor node is assumed to be battery-operated. (It has limited energy supply.) It maintains a sensing interval and a sampling rate for each sensor attached to it. Upon a sensor reading, it stores collected data in its own memory space. Given a data transmission interval, it periodically flushes all data stored in its memory space and transmits the data to a sink node.

Edge Layer: consists of sink nodes, each of which participates in a certain BSN and receives sensor data periodically from sensor nodes in the BSN. A sink node stores incoming sensor data in its memory space and periodically flushes stored data to transmit them to the cloud layer. Different sink nodes have different data transmission intervals. A sink node's data transmission interval can be different from the ones of sensor nodes in the same BSN. Sink nodes are assumed to have limited energy supplies through batteries.

Cloud layer: operates on a cloud(s) that host *virtual sensors*, which are virtualized counterparts (or software counterparts) of physical sensors in BSNs. Virtual sensors collect sensor data from sink nodes in the edge layer and store those data for future use. The cloud layer also hosts various applications that obtain sensor data from virtual sensors and aid medical staff (e.g., clinicians, hospital/visiting nurses and caregivers) to monitor patients and share sensor data for clinical observation and intervention.

BitC performs *push-pull hybrid communication* between its three layers. Each sensor node periodically collects data from a sensor(s) attached to it based on sensor-specific sensing intervals and sampling rates and transmits (or pushes) those collected data to a sink node. The sink node in turn forwards (or pushes) incoming sensor data periodically to virtual sensors in a cloud(s). Cloud applications request sensor data to virtual sensors. If a virtual sensor has requested data, it returns that data. Otherwise, it issues a pull request to a sink node. If the sink node has the requested data in its memory space, it returns that data. Otherwise, it issues another pull request to a sensor node that is responsible for the requested data. Upon receiving a pull request, the sensor node returns the requested data if it has the data in its memory. Otherwise, it returns an error message to a cloud application through a sink node.

While push communication carries out a one-way upstream travel of sensor data, pull communication incurs a round trip for requesting sensor data and receiving that data (or an error message). This push-pull communication is intended to make as much sensor data as possible available for cloud applications by taking advantage of push communication while allowing virtual sensors to pull any missing or extra data anytime in an on-demand manner. For example, when an anomaly is found in pushed sensor data, medical staff may pull extra data in a higher temporal resolution to better understand a patient's medical condition. Given a sufficient amount of data, they may perform clinical intervention, order clinical cares, dispatch ambulances or notify family members of patients.

III. BSN Configuration Problem in BitC

This section describes a BSN configuration problem for which BitC seeks equilibrium solutions. Each BSN configuration consists of four types of parameters (i.e., decision variables): sensing intervals and sampling rates for sensors as well as data transmission intervals for sensor and sink nodes. The problem is stated with the following symbols.

- $B = \{b_1, b_2, ..., b_i, ..., b_N\}$ denotes the set of N BSNs, each of which operates for a patient.
- Each BSN b_i consists of a sink node (denoted by m_i) and M sensor nodes: $b_i = \{h_{i1}, h_{i2}, ..., h_{ij}, ..., h_{iM}\}$. Each sensor node h_{ij} has L sensors: $h_{ij} = \{s_{ij1}, s_{ij2}, ..., s_{ijk}, ..., s_{ijL}\}$. o_{ijk} is the data transmission interval for h_{ij} to transmit sensor data collected from s_{ijk}. p_{ijk} and q_{ijk} are the sensing interval and sampling rate for s_{ijk}. Sampling rate is defined as the number of sensor data samples collected in a unit time. Each sensor node stores collected sensor data in its memory space until its next push transmission. If the memory becomes full, it performs FIFO (First-In-First-Out) data replacement. In a push transmission, it flushes and sends out all data stored in its memory.
- o_{m_i} denotes the data transmission interval for m_i to forward (or push) sensor data incoming from sensor nodes in b_i In between two push transmissions, m_i stores sensor data from b_i in its memory. It performs FIFO data replacement if the memory becomes full. In a push transmission, it flushes and sends out all data stored in the memory.
- $R_{ijk} = \{r_{ijk1}, r_{ijk2}, ..., r_{ijkr}, ..., r_{ijk|R_{ijk}|}\}$ denotes the set of sensor data requests that cloud applications issue to the

virtual counterpart of s_{ijk} (s'_{ijk}) during the time period of W in the past. Each request r_{ijkr} is characterized by its time stamp (t_{ijkr}) and time window (w_{ijkr}). It retrieves all sensor data available in the time interval $[t_{ijkr} - w_{ijkr}, t_{ijkr}]$. If s'_{ijk} has at least one data in the interval, it returns those data; otherwise, it issues a pull request to m_i.

- $R^m_{ijk} \subseteq R_{ijk}$ denotes the set of sensor data requests for which a virtual sensor s'_{ijk} has no data. $|R^m_{ijk}|$ indicates the number of pull requests that s'_{ijk} issues to m_i. In other words, $R_{ijk} \setminus R^m_{ijk}$ is the set of sensor data requests that s'_{ijk} fulfills regarding s_{ijk}.
- $R^s_{ijk} \subseteq R^m_{ijk} \subseteq R_{ijk}$ denotes the set of sensor data requests for which m_i has no data. $|R^s_{ijk}|$ indicates the number of pull requests that m_i issues to h_{ij} for collecting data from s_{ijk}. $R^m_{ijk} \setminus R^s_{ijk}$ is the set of sensor data requests that m_i fulfills regarding s_{ijk}.

This paper considers four performance objectives: bandwidth consumption between the edge and cloud layers (f_B), energy consumption of sensor and sink nodes (f_E), request fulfillment for cloud applications (f_R) and data yield for cloud applications (f_D). The first two objectives are to be minimized while the others are to be maximized.

The bandwidth consumption objective (f_B) is defined as the total amount of data transmitted per a unit time between the edge and cloud layers. This objective impacts the payment for bandwidth consumption based on a cloud operator's pay-per-use billing scheme. It also impacts the lifetime of sink nodes. f_B is computed as follows.

$$\begin{aligned} f_B =& \frac{1}{W}\sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{L}(c_{ijk}d_{ijk}) + \frac{1}{W}\sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{L}\sum_{r=1}^{|R^m_{ijk}|}(\phi_{ijkr}d_{ijk} + d_r) \\ &+ \frac{1}{W}\sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{L}\sum_{r=1}^{|R^s_{ijk}|} e_r(|R^s_{ijk}| - \eta_{ijkr}) \end{aligned} \tag{1}$$

The first and second terms indicate the bandwidth consumption by one-way push communication from the edge layer to the cloud layer and two-way pull communication between the cloud and edge layers, respectively. c_{ijk} denotes the number of sensor data that s_{ijk} generates and sink nodes in turn push to the cloud layer during W. d_{ijk} denotes the size of each sensor data (in bits) that s_{ijk} generates. It is currently computed as: $q_{ijk} \times 16$ bits/sample. ϕ_{ijkr} denotes the number of sensor data that a pull request $r \in R^m_{ijk}$ can collect from sink nodes ($\phi_{ijkr} = |R^m_{ijk} \setminus R^s_{ijk}|$). d_r is the size of a pull request transmitted from the cloud layer to the edge layer. The third term in Eq. 1 indicates the bandwidth consumption by the error messages that sensors generate because they fail to fulfill pull requests. η_{ijkr} is the number of sensor data that a pull request $r \in R^s_{ijk}$ can collect from sensor nodes. e_r is the size of an error message.

The energy consumption objective (f_E) is defined as the total amount of energy that sensor and sink nodes consume for data transmissions during W. It impacts the lifetime of sensor and sink nodes. It is computed as follows.

$$\begin{aligned} f_E =& \sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{L}\frac{W}{o_{ijk}}e_t d_{ijk} + \sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{L}\sum_{r=1}^{|R^s_{ijk}|} e_t\eta_{ijkr}(d_{ijk} + d'_r) \\ &+ \sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{L}\frac{W}{o_{m_i}}e_t d_{ijk} + \sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{L}\sum_{r=1}^{|R^m_{ijk}|} e_t\phi_{ijkr}(d_{ijk} + d_r) \\ &+ 2 \times \sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{L}\sum_{r=1}^{|R^s_{ijk}|} e_t e_r(|R^s_{ijk}| - \eta_{ijkr}) \end{aligned} \tag{2}$$

The first and second terms indicate the energy consumption by one-way push communication from the sensor layer to the edge layer and two-way pull communication between the edge layer and the sensor layer, respectively. e_t denotes the amount of energy (in Watts) that a sensor or sink node consumes to transmit a single bit of data. d'_r denotes the size of a pull request from the edge layer to the sensor layer. The third and fourth terms indicate the energy consumption by push and pull communication between the edge and cloud layer, respectively. The fifth term indicates the energy consumption for transmitting error messages on sensor and sink nodes.

The request fulfillment objective (f_R) is the ratio of the number of fulfilled requests over the total number of requests:

$$f_R = \frac{\sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{L}\sum_{r=1}^{|R_{ijk}|} I_{R_{ijk}}}{|R_{ijk}|} \times 100 \tag{3}$$

$I_{R_{ijk}} = 1$ if a request $r \in R_{ijk}$ obtains at least one sensor data; otherwise, $I_{R_{ijk}} = 0$.

The data yield objective (f_Y) is defined as the total amount of data that cloud applications gather for their users. This objective impacts the informedness and situation awareness for application users. It is computed as follows.

$$f_Y = \sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{L}\sum_{r=1}^{|R^m_{ijk}|}\phi_{ijkr} + \sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{L}\sum_{r=1}^{|R^s_{ijk}|}\eta_{ijkr} + c_{ijk} \tag{4}$$

BitC considers four constraints. The first constraint (C_E) is the upper limit for energy consumption: $f_E < C_E$. A violation for the constraint (g_E) is computed as $g_E = I_E \times (f_E - C_E)$ where $I_E = 1$ if $f_E > C_E$; otherwise $I_E = 0$.

The second constraint (C_Y) is the lower limit for data yield: $f_Y > C_Y$. A constraint violation (g_Y) is computed as $g_Y = I_Y \times (C_Y - f_Y)$ where $I_Y = 1$ if $f_Y < C_Y$; otherwise $I_Y = 0$.

The third constraint (C_R) is the lower limit for request fulfillment: $f_R > C_R$. The constraint violation in request fulfillment (g_R) is computed as $g_R = I_R \times (C_R - f_R)$ where $I_R = 1$ if $f_R < C_R$; otherwise $I_R = 0$.

The fourth constraint (C_B) is the upper limit for bandwidth consumption: $f_B < C_B$. A violation for this constraint (g_B) is computed as $g_B = I_B \times (f_B - C_B)$ where $I_B = 1$ if $f_B > C_B$; otherwise $I_B = 0$.

IV. Background: Evolutionary Game Theory

In a conventional game, the objective of a player is to choose a strategy that maximizes its payoff in a single game. In contrast, evolutionary games are played repeatedly by players randomly drawn from a population [2]. This section overviews key elements in evolutionary games: evolutionarily stable strategies (ESS) and replicator dynamics.

A. Evolutionarily Stable Strategies (ESS)

Suppose all players in the initial population are programmed to play a certain (incumbent) strategy k. Then, let a small population share of players, $x \in (0, 1)$, mutate and play a different (mutant) strategy ℓ. When a player is drawn for a game, the probabilities that its opponent plays k and ℓ are $1 - x$ and x, respectively. Thus, the expected payoffs for the player to play k and ℓ are denoted as $U(k, x\ell + (1-x)k)$ and $U(\ell, x\ell + (1-x)k)$, respectively.

Definition 1. *A strategy k is said to be evolutionarily stable if, for every strategy $\ell \neq k$, a certain $\bar{x} \in (0, 1)$ exists, such that the inequality*

$$U(k,\ x\ell + (1-x)k) > U(\ell,\ x\ell + (1-x)k)$$

holds for all $x \in (0, \bar{x})$.

If the payoff function is linear, Equation 5 derives:

$$(1-x)U(k,k) + xU(k,\ell) > (1-x)U(\ell,k) + xU(\ell,\ell)$$

If x is close to zero, Equation 5 derives either

$$U(k,k) > U(\ell,k) \textit{ or } U(k,k) = U(\ell,k) \textit{ and } U(k,\ell) > U(\ell,\ell)$$

This indicates that a player associated with the strategy k gains a higher payoff than the ones associated with the other strategies. Therefore, no players can benefit by changing their strategies from k to the others. This means that an ESS is a solution on a Nash equilibrium. An ESS is a strategy that cannot be invaded by any alternative (mutant) strategies that have lower population shares.

B. Replicator Dynamics

The replicator dynamics describes how population shares associated with different strategies evolve over time. Let $\lambda_k(t) \geq 0$ be the number of players who play the strategy $k \in K$, where K is the set of available strategies. The total population of players is given by $\lambda(t) = \sum_{k=1}^{|K|} \lambda_k(t)$. Let $x_k(t) = \lambda_k(t)/\lambda(t)$ be the population share of players who play k at time t. The population state is defined by $X(t) = [x_1(t), \cdots, x_k(t), \cdots, x_K(t)]$. Given X, the expected payoff of playing k is denoted by $U(k, X)$. The population's average payoff, which is same as the payoff of a player drawn randomly from the population, is denoted by $U(X, X) = \sum_{k=1}^{|K|} x_k \cdot U(k, X)$. In the replicator dynamics, the dynamics of the population share x_k is described as follows. $\dot{x}_k$ is the time derivative of x_k.

$$\dot{x}_k = x_k \cdot [U(k, X) - U(X, X)] \qquad (5)$$

This equation states that players increase (or decrease) their population shares when their payoffs are higher (or lower) than the population's average payoff.

Theorem 1. *If a strategy k is strictly dominated, then* $x_k(t)_{t\to\infty} \to 0$.

A strategy is said to be strictly dominant if its payoff is strictly higher than any opponents. As its population share grows, it dominates the population over time. Conversely, a strategy is said to be strictly dominated if its payoff is lower than that of a strictly dominant strategy. Thus, strictly dominated strategies disappear in the population over time.

There is a close connection between Nash equilibria and the steady states in the replicator dynamics, in which the population shares do not change over time. Since no players change their strategies on Nash equilibria, every Nash equilibrium is a steady state in the replicator dynamics. As described in Section IV-A, an ESS is a solution on a Nash equilibrium. Thus, an ESS is a solution at a steady state in the replicator dynamics. In other words, an ESS is the strictly dominant strategy in the population on a steady state.

BitC maintains a population of configuration strategies for each BSN. In each population, strategies are randomly drawn to play games repeatedly until the population state reaches a steady state. Then, BitC identifies a strictly dominant strategy in the population and configures a BSN based on the strategy as an ESS.

V. Body-in-the-Cloud

BitC maintains N populations, $\{\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_N\}$, for N BSNs and performs games among strategies in each population. Each strategy $s(b_i)$ specifies a particular configuration for a BSN b_i using four types of parameters: sensing intervals and sampling rates for sensors (p_{ij} and q_{ij}) as well as data transmission intervals for sink and sensor nodes (o_{m_i} and o_{ij}).

$$s(b_i) = \bigcup_{j \in 1..M} (o_{m_i}, o_{ij}, p_{ij}, q_{ij}) \quad 1 < i < N \qquad (6)$$

Algorithm 1 shows how BitC seeks an evolutionarily stable configuration strategy for each BSN through evolutionary games. In the 0-th generation, strategies are randomly generated for each of N populations $\{\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_N\}$ (Line 2). Those strategies may or may not be feasible. Note that a strategy is said to be feasible if it violates none of four constraints described in Section III.

In each generation (g), a series of games are carried out on every population (Lines 4 to 27). A single game randomly chooses a pair of strategies (s_1 and s_2) and distinguishes them to the winner and the loser with respect to performance objectives described in Section III (Lines 7 to 9). The winner is replicated to increase its population share and mutated with polynomial mutation (Lines 10 to 18) [4]. Mutation randomly chooses a parameter (or parameters) in a given strategy with a certain mutation rate P_m and alters its/their value(s) at random (Lines 12 to 14). Then the loser of the game is replaced by the winner's replica (Line 17). Once all strategies play games in the population, BitC identifies a feasible strategy whose population share (x_s) is the highest and determines it as a dominant strategy (d_i) (Lines 20 to 24). In the end, BitC uses the dominant strategy to adjust the configuration parameters for a BSN in question (Line 25).

A game is carried out based on the superior-inferior relationship between given two strategies and their feasibility (c.f. `performGame()` in Algorithm 1). If a feasible strategy and an infeasible strategy participate in a game, the feasible one always wins over its opponent. If both strategies are feasible, they are compared with their hypervolume value.

Hypervolume (HV) metric [5] measures the volume that a given strategy s dominates in the objective space:

$$HV(s) = \Lambda\left(\bigcup\{x'|s \succ x' \succ x_r\}\right) \quad (7)$$

Λ denotes the Lebesgue measure. x_r is the reference point placed in the objective space. A higher hypervolume means that a strategy is more optimal. Given two strategies, the one with a higher hypervolume value wins a game. If both have the same hypervolume value, the winner is randomly selected.

If both strategies are infeasible in a game, they are compared based on their constraint violation. An infeasible strategy s_1 wins a game over another infeasible strategy s_2 if both of the following conditions hold:

- s_1's constraint violation is lower than, or equal to, s_2's in all constraints.
- s_1's constraint violation is lower than s_2's in at least one constraints.

Algorithm 1 Evolutionary Process in BitC

```
1: $g$ = 0
2: Randomly generate the initial $N$ populations for $N$ BSNs: $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_N\}$
3: while $g < G_{max}$ do
4:     for each population $\mathcal{P}_i$ randomly selected from $\mathcal{P}$ do
5:         $\mathcal{P}'_i \leftarrow \emptyset$
6:         for $j = 1$ to $|\mathcal{P}_i|/2$ do
7:             $s_1 \leftarrow$ randomlySelect($\mathcal{P}_i$)
8:             $s_2 \leftarrow$ randomlySelect($\mathcal{P}_i$)
9:             $\{winner, loser\} \leftarrow$ performGame($s_1$, $s_2$)
10:            $replica \leftarrow$ replicate($winner$)
11:            for each parameter $v$ in $replica$ do
12:                if random() $\leq P_m$ then
13:                    $replica \leftarrow$ mutate($replica$, $v$)
14:                end if
15:            end for
16:            $\mathcal{P}_i \setminus \{s_1, s_2\}$
17:            $\mathcal{P}'_i \cup \{winner, replica\}$
18:        end for
19:        $\mathcal{P}_i \leftarrow \mathcal{P}'_i$
20:        $d_i \leftarrow argmax_{s \in \mathcal{P}_i} x_s$
21:        while $d_i$ is infeasible do
22:            $\mathcal{P}_i \setminus \{d_i\}$
23:            $d_i \leftarrow argmax_{s \in \mathcal{P}_i} x_s$
24:        end while
25:        Use $d_i$ to adjust the parameters for a BSN in question.
26:    end for
27:    $g = g + 1$
28: end while
```

VI. Stability Analysis

This section analyzes BitC's stability (i.e., reachability to at least one of Nash equilibrium) by proving the state of each population converges to an evolutionarily stable equillibrium. The proof consists of three steps: (1) designing a set of differential equations that describe the dynamics of the population state, (2) proving an strategy selection process has equilibria, and (3) proving the the equilibria are asymptotically (or evolutionarily) stable. The proof uses the following symbols:

- S denotes the set of available strategies. S^* denotes a set of strategies that appear in the population.
- $X(t) = \{x_1(t), x_2(t), \cdots, x_{|S^*|}(t)\}$ denotes a population state at time t where $x_s(t)$ is the population share of a strategy $s \in S$. $\sum_{s \in S^*}(x_s) = 1$.
- F_s denotes the fitness of a strategy s. It is a relative value that is determined in a game against an opponent based on the dominance relationship between them (Algorithm **??**). The winner of a game earns a higher fitness than the loser.
- $p^s_k = x_k \cdot \phi(F_s - F_k)$ denotes the probability that a strategy s is replicated by winning a game against another strategy k. $\phi(F_s - F_k)$ is the probability that the fitness of s is higher than that of k.

The dynamics of the population share of s is described as:

$$\begin{aligned}\dot{x}_s &= \sum_{k \in S^*, k \neq s} \{x_s p^s_k - x_k p^k_s\} \\ &= x_s \sum_{k \in S^*, k \neq s} x_k\{\phi(F_s - F_k) - \phi(F_k - F_s)\} \quad (8)\end{aligned}$$

Note that if s is strictly dominated, $x_s(t)_{t \to \infty} \to 0$.

Theorem 2. *The state of a population converges to an equilibrium.*

Proof. It is true that different strategies have different fitness values. In other words, only one strategy has the highest fitness among others. Given Theorem 1, assuming that $F_1 > F_2 > \cdots > F_{|S^*|}$, the population state converges to an equilibrium: $X(t)_{t \to \infty} = \{x_1(t), x_2(t), \cdots, x_{|S^*|}(t)\}_{t \to \infty}$ $= \{1, 0, \cdots, 0\}$. □

Theorem 3. *The equilibrium found in Theorem 2 is asymptotically stable.*

Proof. At the equilibrium $X = \{1, 0, \cdots, 0\}$, a set of differential equations can be downsized by substituting $x_1 = 1 - x_2 - \cdots - x_{|S^*|}$

$$\dot{z}_s = z_s[c_{s1}(1 - z_s) + \sum_{i=2, i \neq s}^{|s^*|} z_i \cdot c_{si}], \quad s, k = 2, ..., |S^*| \quad (9)$$

where $c_{sk} \equiv \phi(F_s - F_k) - \phi(F_k - F_s)$ and $Z(t) = \{z_2(t), z_3(t), \cdots, z_{|S^*|}(t)\}$ denotes the corresponding downsized population state. Given Theorem 1, $Z_{t \to \infty} = Z_{eq} = \{0, 0, \cdots, 0\}$ of $(|S^*| - 1)$-dimension.

If all Eigenvalues of Jaccobian matrix of $Z(t)$ has negative real parts, Z_{eq} is asymptotically stable. The Jaccobian matrix J's elements are described as follows where $s, k = 2, ..., |S^*|$.

$$J_{sk} = \left[\frac{\partial \dot{z}_s}{\partial z_k}\right]_{|Z = Z_{eq}} = \left[\frac{\partial z_s[c_{s1}(1 - z_s) + \sum_{i=2, i \neq s}^{|S^*|} z_i \cdot c_{si}]}{\partial z_k}\right]_{|Z = Z_{eq}} \quad (10)$$

Therefore, J is given as follows, where $c_{21}, c_{31}, \cdots, c_{|S^*|1}$ are J's Eigenvalues.

$$J = \begin{bmatrix} c_{21} & 0 & \cdots & 0 \\ 0 & c_{31} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & c_{|S^*|1} \end{bmatrix} \quad (11)$$

$c_{s1} = -\phi(F_1 - F_s) < 0$ for all s; therefore, $Z_{eq} = \{0, 0, \cdots, 0\}$ is asymptotically stable. □

VII. Simulation Evaluation

This section evaluates BitC through simulations and studies how BitC adapts BSN configurations to given operational conditions (e.g., data request patterns placed by cloud applications and memory space availability in sink and sensor nodes).

Simulations are configured with the parameters shown in Table I. Data requests are uniformly distributed over virtual sensors. A time window is randomly set for each request to a sensor. Mutation rate is set to 1/V where V is the number of parameters in a strategy. The following four constraints are used: $C_E = \infty$, $C_Y = 1900$, $C_R = 97$, $C_B = 10$. Every simulation result is the average with 10 independent simulation runs. BitC is evaluated in comparison with NSGA-III, which is one of the state-of-the-art EMOAs [3]. BitC and NSGA-III use the same parameter settings shown in Table I. All other NSGA-III settings are borrowed from [3].

TABLE I: Simulation Settings

Parameter	Value
Duration of a simulation (W)	10,800 secs (3 hrs)
Number of simulation runs	10
Number of BSNs (N)	100
Number of sensor nodes in a BSN (M)	4
Memory space in a sensor node	2 GB
Memory space in a sink node	16 GB
Total number of data requests from cloud apps	1,000
Size of a data request (d_r and d'_r)	100 bytes
Size of an error message (e_r)	250 bytes
Energy consumption for a single bit of data (e_t)	0.001 Watt
Blood pressure request time window	[0, 1000 secs]
Acelerometer request time window	[0, 1800 secs]
ECG request time window	[0, 600 secs]
Number of generations (G_{max})	100
Mutation rate (P_m)	1/V

Table II examines how a mutation-related parameter, called distribution index (η_m in [4]), impacts the performance of BitC. This parameter controls how likely a mutated strategy is similar to its original. (A higher distribution index makes a mutant more similar to its original.) In Table II, the performance of BitC is evaluated with the hypervolume measure that a set of dominant strategies yield in the 100th generation. The hypervolume metric indicates the union of the volumes that a given set of solutions dominates in the objective space [5]. A higher hypervolume means that a set of solutions is more optimal. As shown in Table II, BitC yields the best performance with the distribution index value of 30. (Each each population ($|\mathcal{P}_i|$ in Algorithm 1) contains 100 strategies.) Thus, this parameter setting is used in all successive simulations.

TABLE II: Impacts of Distribution Index Values on Hypervolume

Dist. Index	Hypervolume	Dist. Index	Hypervolume
20	0.922	25	0.931
30	0.949	35	0.942
40	0.934		

Table III examines how the size of each population ($|\mathcal{P}_i|$ in Algorithm 1) impacts the hypervolume performance of BitC. The highest hypervolume result is obtained with the population size of 100; however, it is close enough to the result with the population size of 50. Therefore, the population size is set to 50 in all successive simulations in favor of reducing the execution time of BitC's optimization process.

TABLE III: Impacts of Population Size on Hypervolume (HV)

Population Size	Hypervolume
20	0.900
50	0.928
100	0.929

Table IV compares BitC-HV and NSGA-III based on three metrics: objective values, hypervolume and Euclidean distance. For NSGA-III, objective values are measured with an individual that minimizes the Euclidean distance to the BitC solution. Hypervolume is measured with the NSGA-III individual. Distance is computed with each objective normalized to [0, 1]. (The value range of a distance is [0, 2].)

As shown in Table IV, BitC is non-dominated (or tie) with NSGA-III with respect to four objective values. BitC outperforms NSGA-III in three objectives (request fulfillment, bandwidth consumption and energy consumption) while NSGA-III outperforms BitC in data yield. BitC yields a slightly higher (1.5% higher) hypervolume value than NSGA-III. Table IV examines the distances in the normalized objective space from the Utopian point, (0, 0, 0, 0), to the BitC and NSGA-III solutions. Euclidean and Manhattan distance metrics are used. In both metrics, a shorter distance means that a given solution is closer to the Utopian point (i.e., more optimal). BitC is closer to the Utopian point than NSGA-III by 49.48% and 57.58% in Manhattan and Euclidean distances, respectively. Table IV demonstrates that BitC-HV outperforms NSGA-III.

TABLE IV: Comparison of BitC and NSGA-III

Objective	BitC-HV	NSGA-III
Request Fulfillment: f_R (%)	98.5	98
Bandwidth: f_B (Kbps)	3.12	9.99
Energy consumption: f_E (Watts)	56.25	172.36
Data Yield: f_Y	8025	46252
Hypervolume (HV)	0.948	0.924
Euclidean distance to the Utopian point	0.271	0.639
Manhattan distance to the Utopian point	0.538	1.065
# of generations to reach HV=0.924	19	133
# of generations to reach HV=0.948	96	—

The bottom two rows of Table IV show BitC's and NSGA-III's convergence speed. NSGA-III requires 133 generations to reach the HV value of 0.924, which is the highest HV that NSGA-III yields. In contrast, BitC spends only 19 generations to reach the HV value. It maintains a 7x speedup against NSGA-III in convergence speed.

Table V shows the variance of objective values that BitC-HV and NSGA-III yield at the last generation in 10 different simulation runs. A lower variance means higher stability (or higher similarity) in objective value results (i.e., lower oscillations in objective value results) among different simulation runs. BitC-HV maintains significantly higher stability than NSGA-III in all objectives. On average, BitC's stability is 41.75% higher than NSGA-III's. This result exhibits BitC's stability property (i.e. reachability to at least one equillibira), which NSGA-III does not have.

TABLE V: Stability of Objective Values in BitC and NSGA-III

Objectives	BitC-HV	NSGA-III	Diff (%)
Request Fulfillment: f_R	0.04	0.05	20%
Bandwidth: f_B	0.02	0.11	81.81%
Energy Consumption: f_E	0.017	0.05	66%
Data Yield: f_Y	0.01	0.01	0%
Average Difference (%)	–	–	41.75%

Table VI compares the execution time for BitC and NSGA-III to run a single generation and an entire simulation (i.e., 100 generations). It illustrates that BitC's execution time is 12.4% shorter than NSGA-III's.

TABLE VI: Execution time of BitC and NSGA-III

	BitC	NSGA-III
Per-generation execution time	10.61 sec	12.11sec
Total execution time	2.95 hrs	3.37 hrs

In summary, Tables IV to VI demonstrate that BitC outperforms NSGA-III in optimality, stability and efficiency.

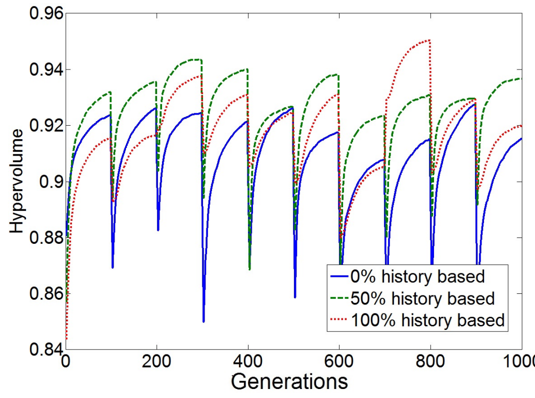


Fig. 2: History-based Optimization under Dynamic Changes in Operational Conditions

Fig. 2 shows how BitC reconfigures BSNs when operational conditions change dynamically. Upon a dynamic change, BitC reruns its optimization process by mutating the dominant strategies obtained in the previous run through polynomial mutation (c.f. Lines 11 to 15 in Algorithm 1) and supplying mutants to the initial population for the current run. Fig. 2 shows how the number of those mutants impacts the HV performance when the number of patients and the total number of data requests increases by 10%. BitC runs 10 times; it executes 1,000 generations in total. "100% history-based" means all the initial strategies are mutants produced from the dominant strategies obtained in the previous run. "50% history-based" means that a half of them are mutants and the other half is randomly generated. "0% history-based" means all are randomly generated. In the 50% case, BitC often converges faster and reaches a higher HV compared to the other two cases.

Table VII depicts the average convergence speed and optimality in 10 consecutive simulation runs. In 50% case, BitC requires 10 generations to reach the HV value of 0.91 while it requires 43 generations in the 0% case. This "50% history-based" optimization gains 4.3x speedup and 1.4% higher HV compared to the case where no history is used. Thus, this parameter setting is used in successive simulations.

TABLE VII: Impacts of History-based Optimization on Convergence Speed and Optimality

	History-level		
	0%	50%	100%
# of generations to reach HV=0.91	43	10	29
# of generations to reach HV=0.92	77	21	56
# of generations to reach HV=0.93	-	60	81
Hypervolume (HV) value	0.920	0.933	0.925

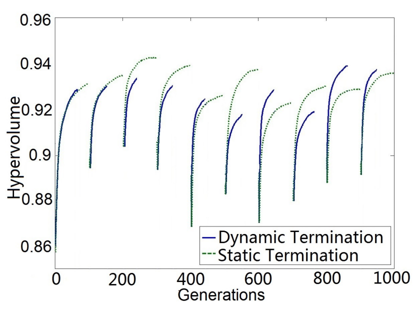
Fig. 3 illustrates how BitC's optimization results change if it introduces a dynamic termination condition. Unlike a static termination condition used in the previous simulations, which stops an optimization process when the number of generations reaches 100, the dynamic termination condition stops an optimization process when HV does not increase by 0.1% in the past 10 generations. Fig. 3a shows the impacts of dynamic termination on HV performance under no changes in operational conditions. Fig. 3b and Fig. 3c show the impacts of dynamic termination under dynamic 10% and 50% changes operational conditions, respectively. In each change, the number of patients and the total number of data requests increase or decrease at random. As illustrated in Table VIII, dynamic termination gains 1.8x speedup in convergence speed while sacrificing 0.3% in HV when no operational conditions change. Under 10% and 50% changes in operational conditions, dynamic termination gains 1.7x and 1.8x speedup in convergence speed, respectively, while sacrificing 0.1% and 0.3% in HV. Fig. 3 and Table VIII demonstrate that dynamic optimization termination improves convergence speed and makes virtually no sacrifices on HV performance.

TABLE VIII: Impacts of Dynamic Optimization Termination on Convergence Speed and Optimality

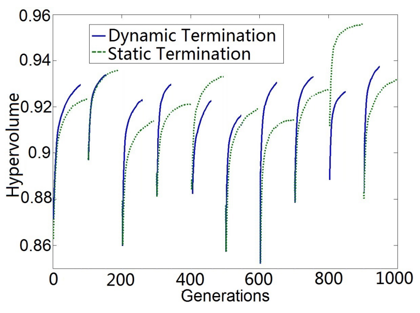
		# of generations	HV
No changes	Static termination	100	0.933
	Dynamic termination	56	0.930
±10% changes	Static termination	100	0.927
	Dynamic termination	59	0.928
±50% changes	Static termination	100	0.929
	Dynamic termination	56	0.923

VIII. Related Work

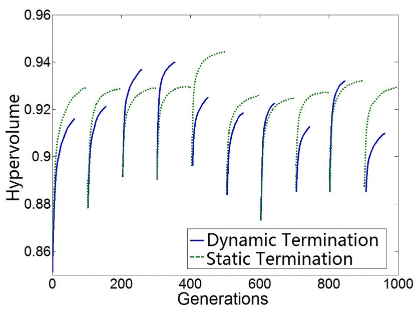
Various architectures and research tools have been proposed for cloud-integrated sensor networks including BSNs [6]–[19]. Many of them, [6]–[14], assume three-tier architectures similar to BitC and investigate publish/subscribe communication between the edge layer to the cloud layer. Their focus is placed on push communication. In contrast, BitC investigates push-pull hybrid communication between the sensor layer and the cloud layer through the edge layer. Yuriyama et al. [15], Rollin et al. [16] and Chung et al. [18] propose a two-tier architecture that consists of the sensor and cloud



(a) No Changes



(b) $\pm 10\%$ Changes



(c) $\pm 50\%$ Changes

Fig. 3: Impacts of a Dynamic Optimization Termination on Hypervolume Performance

layers. The architectures proposed by Yuriyama et al. and Fortino et al. [19] are similar to BitC in that they leverage the notion of virtual sensors. However, they do not consider push-pull (nor publish/subscribe) communication. All the above-mentioned relevant work do not consider adaptive/stable configurations of sensor networks as BitC does [6]–[19].

Push-pull hybrid communication has been studied in sensor networks [20]–[23]. However, few efforts exist to study it between the edge and cloud layers in the context of cloud-integrated sensor networks. Unlike those relevant work, this paper formulates a sensor network configuration problem with cloud-specific objectives as well as the ones in sensor networks and seeks adaptive/stable solutions for the problem.

IX. Conclusion

This paper considers a layered push-pull hybrid communication for cloud-integrated BSNs and formulates a BSN configuration problem to seek adaptive and stable solutions. An evolutionary game theoretic algorithm is used to approach the problem. A theoretical analysis proves that the proposed algorithm allows each BSN to operate at an equilibrium by using an evolutionarily stable configuration strategy in a deterministic (i.e., stable) manner. Simulation results verify this theoretical analysis; BSNs seek equilibria to perform adaptive and evolutionarily stable configuration strategies.

References

[1] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons Inc, 2001.

[2] J. W. Weibull, *Evolutionary Game Theory*. MIT Press, 1996.

[3] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints," *IEEE Trans. Evol. Computat.*, vol. 18, no. 4, 2014.

[4] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans Evol. Computat.*, vol. 6, no. 2, 2002.

[5] E. Zitzler and L. Thiele, "Multiobjective optimization using evolutionary algorithms: A comparative study," in *Proc. Int'l Conf. on Parallel Problem Solving from Nature*, 1998.

[6] M. M. Hassan, B. Song, and E.-N. Huh, "A framework of sensor-cloud integration opportunities and challenges," in *Proc. the 3rd ACM Int'l Conference on Ubiquitous Info. Mgt. and Comm.*, 2009.

[7] K. Aberer, M. Hauswirth, and A. Salehi, "Infrastructure for data processing in large-scale interconnected sensor networks," in *Proc. the 8th IEEE Int'l Conference on Mobile Data Management*, 2007.

[8] M. Gaynor, M. Welsh, S. Moulton, A. Rowan, E. LaCombe, and J. Wynne, "Integrating wireless sensor networks with the grid," *IEEE Internet Computing*, July/August 2004.

[9] P. Boonma and J. Suzuki, "TinyDDS: An interoperable and configurable publish/subscribe middleware for wireless sensor networks," in *Principles and Apps. of Dist. Event-Based Systems*, A. Hinze and A. Buchmann, Eds. IGI Global, 2010, ch. 9.

[10] N. Suryadevara, M. T. Quazi, and S. Mukhopadhyay, "Intelligent sensing systems for measuring wellness indices of the daily activities for the elderly," in *Proc. Int'l Conf. on Intelligent Environments*, 2012.

[11] A. Ambrose, M. Cardei, and I. Cardei, "Patient-centric hurricane evacuation management system," in *Proc. IEEE Int'l Performance Computing and Communications Conference*, 2010.

[12] M. Boulmalf, A. Belgana, T. Sadiki, S. Hussein, T. Aouam, and H. Harroud, "A lightweight middleware for an e-health wsn based system using android technology," in *Proc. Int'l Conf. on Multimedia Comp. and Sys.*, 2012.

[13] K. E. U. Ahmed and M. A. Gregory, "Integrating wireless sensor networks with cloud computing," in *Proc. Int'l Conf. on Mobile Ad-hoc and Sensor Networks*, 2011.

[14] P. Zhang, Z. Yan, and H. Sun, "A novel architecture based on cloud computing for wireless sensor network," in *Proc. Int'l Conf. on Computer Science and Electronics Engineering*, 2013.

[15] M. Yuriyama and T. Kushida, "Sensor-cloud infrastructure - physical sensor management with virtualized sensors on cloud computing," in *Proc. the 13th Int'l Conf. on Network-Based Info. Sys.*, 2010.

[16] C. O. Rolim, F. L. Koch, C. B. Westphall, J. Werner, A. Fracalossi, and G. S. Salvador, "A cloud computing solution for patient's data collection in health care institutions," in *Proc. the 2nd IARIA Int'l Conference on eHealth, Telemedicine and Social Medicine*, 2010.

[17] N. B. Priyantha, A. Kansal, M. Goraczko, and F. Zhao, "Tiny web services: design and implementation of interoperable and evolvable sensor networks," in *Proc. the 6th ACM Int'l Conference on Embedded Network Sensor Systems*, 2008.

[18] W.-Y. Chung, P.-S. Yu, and C.-J. Huang, "Cloud computing system based on wireless sensor network," in *Proc. Federated Conf. on Comp. Sci. and Info. Sys.*, 2013.

[19] G. Fortino, D. Parisi, V. Pirrone, and G. D. Fatta, "BodyCloud: A SaaS approach for community body sensor networks," *Future Generation Computer Systems*, vol. 35, no. 6, pp. 62–79, 2014.

[20] H. Wada, P. Boonma, and J. Suzuki, "Chronus: A spatiotemporal macroprogramming language for autonomic wireless sensor networks," in *Autonomic Network Mgt. Principles: From Concepts to Applications*, N. Agoulmine, Ed. Elsevier, 2010, ch. 8.

[21] P. Boonma, Q. Han, and J. Suzuki, "Leveraging biologically-inspired mobile agents supporting composite needs of reliability and timeliness in sensor applications," in *Proc. IEEE Int'l Conf. on Frontiers in the Convergence of Biosci. and Info. Tech.*, 2007.

[22] S. Kapadia and B. Krishnamachari, "Comparative analysis of push-pull query strategies for wireless sensor networks," in *Proc. International Conference on Distributed Computing in Sensor Systems*, 2006.

[23] M. Li, D. Ganesan, and P. Shenoy, "PRESTO: Feedback-driven data management in sensor networks," in *Proc. USENIX Symposium on Networked Systems Design and Implementation*, 2006.